

DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUU	UUU	GGGGGGGGGGGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDD	DDD	BBB	UUU	UUU	GGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG
DDDDDDDDDDDD	EEEEEEEEEEEEEE	BBBBBBBBBBBBBB	UUUUUUUUUUUUUU	UUUUUUUUUUUUUU	GGGGGGGGGG

[illegible]

```
1 0001 0 MODULE DBGVALUES(IDENT = 'V04-000') =
2 0002 1 BEGIN
3 0003 1
4 0004 1
5 0005 1
6 0006 1
7 0007 1
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
10 0010 1 * ALL RIGHTS RESERVED.
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
17 0017 1 * TRANSFERRED.
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
21 0021 1 * CORPORATION.
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
25 0025 1 *
26 0026 1 *****
27 0027 1
28 0028 1
29 0029 1 ++
30 0030 1
31 0031 1 FACILITY: VAX-11 DEBUG
32 0032 1
33 0033 1 ABSTRACT:
34 0034 1
35 0035 1 Language-Independent Value Descriptor support routines
36 0036 1
37 0037 1 ENVIRONMENT: VAX/VMS user mode
38 0038 1
39 0039 1 AUTHOR: J. Francis, CREATION DATE: 19-Apr-1982
40 0040 1
41 0041 1 MODIFIED BY:
42 0042 1
43 0043 1 001 WC3 21-Jun-83
44 0044 1 Add support for /PACKED and /DATE_TIME
45 0045 1
46 0046 1 002 WC3 15-Jul-83
47 0047 1 Fix /DATE_TIME to use DBG$CVT_DX_DX
48 0048 1
49 0049 1 003 WC3 15-Sep-83
50 0050 1 Update DBG$GL_CURRENT_PRIMARY for self-referential records
51 0051 1
52 0052 1 004 WC3 22-Sep-83
53 0053 1 Check for variant records that have been optomized away
54 0054 1 but the DST is still around.
55 0055 1 !--
```



```
.. 57      0056 1  :
.. 58      0057 1  : Table of Contents
.. 59      0058 1  :
.. 60      0059 1  : FORWARD ROUTINE
.. 61      0060 1  : Global Routines :
.. 62      0061 1  :   dbg$data_length,
.. 63      0062 1  :   dbg$make_skeleton_desc,
.. 64      0063 1  :   dbg$make_integer_desc,
.. 65      0064 1  :   dbg$fill_in_vms_desc,
.. 66      0065 1  :   dbg$make_val_desc,
.. 67      0066 1  :   dbg$make_vms_desc,
.. 68      0067 1  :   dbg$prim_to_val,
.. 69      0068 1  :   dbg$print_aggregate      : NOVALUE,
.. 70      0069 1  :   dbg$print_value         : NOVALUE,
.. 71      0070 1  :   dbg$print_value_as_integer : NOVALUE,
.. 72      0071 1  :   dbg$print_vms_value      : NOVALUE;

: fill in vms descriptor
: Create value descriptor
: Create VAX/VMS descriptor
: Get value of a primary
: Print aggregate value
: Print value from DEBUG descriptor
: Print integer in given radix
: Print value from VMS descriptor
```

```
74 0072 1 |
75 0073 1 | INCLUDE FILES
76 0074 1 |
77 0075 1 | REQUIRE 'SRC$:DBGPROLOG';
78 0209 1 |
79 0210 1 |
80 0211 1 | EXTERNALS
81 0212 1 |
82 0213 1 | EXTERNAL
83 0214 1 |     dbg$gl_current_primary,
84 0215 1 |     dbg$gb_radix          : VECTOR[3, BYTE],
85 0216 1 |     dbg$gb_language      : BYTE,
86 0217 1 |     dbg$gv_control        : dbg$control_flags,
87 0218 1 |     dbg$gl_call_context,
88 0219 1 |     dbg$gl_convert_token,
89 0220 1 |     dbg$gl_dflttyp,
90 0221 1 |     dbg$gw_dfltleng       : WORD,
91 0222 1 |     dbg$gl_sign_flag;
92 0223 1 |
93 0224 1 |
94 0225 1 | EXTERNAL ROUTINE
95 0226 1 |     dbg$build_primary_subnode : NOVALUE,
96 0227 1 |     dbg$collect               : NOVALUE,
97 0228 1 |     dbg$cvd_dx_dx             : NOVALUE,
98 0229 1 |     dbg$enum_pos,
99 0230 1 |     dbg$enum_succ,
100 0231 1 |     dbg$enum_val,
101 0232 1 |     dbg$get_tempmem,
102 0233 1 |     dbg$is_tt_entry,
103 0234 1 |     dbg$ins_decode,
104 0235 1 |     dbg$language_format,
105 0236 1 |     dbg$newline               : NOVALUE,
106 0237 1 |     dbg$ngget_radix,
107 0238 1 |     dbg$print                 : NOVALUE,
108 0239 1 |     dbg$print_control         : NOVALUE,
109 0240 1 |     dbg$print_identifier,
110 0241 1 |     dbg$print_set_value       : NOVALUE,
111 0242 1 |     dbg$print_symbol_name     : NOVALUE,
112 0243 1 |     dbg$push_tempmem,
113 0244 1 |     dbg$pop_tempmem           : NOVALUE,
114 0245 1 |     dbg$save_val              : NOVALUE,
115 0246 1 |     dbg$sta_setcontext        : NOVALUE,
116 0247 1 |     dbg$sta_sympathname       : NOVALUE,
117 0248 1 |     dbg$sta_symkind           : NOVALUE,
118 0249 1 |     dbg$sta_symname           : NOVALUE,
119 0250 1 |     dbg$sta_symsize           : NOVALUE,
120 0251 1 |     dbg$sta_syntype           : NOVALUE,
121 0252 1 |     dbg$sta_synvalue          : NOVALUE,
122 0253 1 |     dbg$sta_typefcode,
123 0254 1 |     dbg$sta_typ_atomic        : NOVALUE,
124 0255 1 |     dbg$sta_typ_descr         : NOVALUE,
125 0256 1 |     dbg$sta_typ_enum          : NOVALUE,
126 0257 1 |     dbg$sta_typ_record        : NOVALUE,
127 0258 1 |     dbg$sta_typ_subrng        : NOVALUE,
128 0259 1 |     dbg$sta_typ_typedptr      : NOVALUE,
129 0260 1 |     dbg$sta_variant_value,
130 0261 1 |     dbg$sta_variant_select,
```

Ponter to the primary being processed
Radix settings
Current language
DEBUG status information
Context for 'Bound' values
"Integerize" operator token
Default type from "SET TYPE"
Length of default data-type
Print '+' before the signed variable.

Add sub-node to primary
Sanitize character vectors
Convert data-types by descriptor
Convert value->pos for enum type
Find successor of enum type
Convert pos->value for enum type
Storage space allocator
Check for CALL entry-mask address
Print an instruction
Language override output
Print buffer contents
Obtain radix
Print under FAO format
Control print format
Print name of data item
Print value of set
Print name from a SYMID
Mark current position
Release marked storage
Save value for %CURVAL
Establish RST context
Get fully-qualified data name
Get kind of data item
Get name of data item
Get length of data item
Get type of data item
Get address of data item
Get FCODE of data item
Get symbol table information
Get symbol table information
Get symbol table information
Get symbol table information
Get symbol table information
Get symbol table information
Check value of Variant Tag
Get variant entry (by tag)

```
.. 131      0262 1      for$cvd_d_tg,      ! Conversion routine
.. 132      0263 1      for$cvd_g_tg,      ! Conversion routine
.. 133      0264 1      for$cvd_h_tg;      ! Conversion routine
.. 134      0265 1
.. 135      0266 1
.. 136      0267 1      OWN
.. 137      0268 1      signed dtype      BITVECTOR[dbg$maximum_dtype+1] PRESET(
.. 138      0269 1      [dsc$dtype_f] = 1,
.. 139      0270 1      [dsc$dtype_d] = 1,
.. 140      0271 1      [dsc$dtype_g] = 1,
.. 141      0272 1      [dsc$dtype_h] = 1,
.. 142      0273 1      [dsc$dtype_b] = 1,
.. 143      0274 1      [dsc$dtype_w] = 1,
.. 144      0275 1      [dsc$dtype_l] = 1,
.. 145      0276 1      [dsc$dtype_q] = 1,
.. 146      0277 1      [dsc$dtype_o] = 1,
.. 147      0278 1      [dsc$dtype_p] = 1,
.. 148      0279 1      [dsc$dtype_nz] = 1,
.. 149      0280 1      [dsc$dtype_nl] = 1,
.. 150      0281 1      [dsc$dtype_nlo] = 1,
.. 151      0282 1      [dsc$dtype_nr] = 1,
.. 152      0283 1      [dsc$dtype_nro] = 1,
.. 153      0284 1      [dsc$dtype_sv] = 1,
.. 154      0285 1      [dsc$dtype_svu] = 1);
.. 155      0286 1
.. 156      0287 1      BIND
.. 157      0288 1      Format_AC = UPLIT BYTE(%ASCIC 'AC'),
.. 158      0289 1      Format_AD = UPLIT BYTE(%ASCIC 'AD');
```



```
160 0290 1 GLOBAL ROUTINE DBG$DATA_LENGTH (vms_desc : REF dbg$stg_desc) =
161 0291 1
162 0292 1 FUNCTION
163 0293 1     Given a VMS descriptor, this routine returns the length in
164 0294 1     bits of the object described by the descriptor.
165 0295 1
166 0296 1     Note - for array descriptors, this routine returns the length
167 0297 1     in bits of an element of the array. Do not change it to
168 0298 1     return the length of the entire array - things will break.
169 0299 1
170 0300 1 BEGIN
171 0301 1 LOCAL length;
172 0302 1
173 0303 1 ! Obtain the length from the descriptor. We do not yet know whether
174 0304 1 ! this length is in bits, nibbles, or bytes.
175 0305 1
176 0306 1 length = .vms_desc[dsc$b_length];
177 0307 1
178 0308 1 ! Decide whether the length is in bits, nibbles, or bytes, based
179 0309 1 ! on the dtype.
180 0310 1
181 0311 1 IF .vms_desc[dsc$b_dtype] EQL dsc$sk_bool
182 0312 1 THEN
183 0313 1     length = .length * 1
184 0314 1
185 0315 1 ELSE CASE .vms_desc[dsc$b_dtype] FROM dbg$sk_minimum_dtype TO dbg$sk_maximum_dtype OF
186 0316 1 SET
187 0317 1 [dsc$sk_dtype_f ,dsc$sk_dtype_fc ,dsc$sk_dtype_d ,dsc$sk_dtype_dc ,
188 0318 1 dsc$sk_dtype_g ,dsc$sk_dtype_gc ,dsc$sk_dtype_h ,dsc$sk_dtype_hc ,
189 0319 1 dsc$sk_dtype_b ,dsc$sk_dtype_bu ,dsc$sk_dtype_w ,dsc$sk_dtype_wu ,
190 0320 1 dsc$sk_dtype_l ,dsc$sk_dtype_lu ,dsc$sk_dtype_q ,dsc$sk_dtype_qu ,
191 0321 1 dsc$sk_dtype_o ,dsc$sk_dtype_ou ,dsc$sk_dtype_t ,dsc$sk_dtype_z ,
192 0322 1 dsc$sk_dtype_nl ,dsc$sk_dtype_nlo,dsc$sk_dtype_nr ,dsc$sk_dtype_nro,
193 0323 1 dsc$sk_dtype_nu ,dsc$sk_dtype_nz ,dsc$sk_dtype_zi ,dsc$sk_dtype_zem,
194 0324 1 dsc$sk_dtype_dsc,dsc$sk_dtype_bpv,dsc$sk_dtype_blv,dsc$sk_dtype_adt]: ! M002
195 0325 1
196 0326 1     length = .length * %BPUNIT;
197 0327 1
198 0328 1 [dsc$sk_dtype_vt] :     length = (.length + 2) * %BPUNIT;
199 0329 1
200 0330 1 [dsc$sk_dtype_p] :     length = (.length/2 + 1) * %BPUNIT;
201 0331 1
202 0332 1 [dsc$sk_dtype_tf,dsc$sk_dtype_vu,dsc$sk_dtype_svu] :     0;
203 0333 1
204 0334 1 [dsc$sk_dtype_v ,dsc$sk_dtype_sv] :     IF (.length EQL 0) AND
205 0335 1     (.vms_desc[dsc$b_class] EQL dsc$sk_class_z) THEN
206 0336 1     length = .vms_desc[dsc$l_pos];
207 0337 1
208 0338 1 [dsc$sk_dtype_ac] :     length = (1 +
209 0339 1     .(.vms_desc[dsc$a_pointer])<0,8,0>) * %BPUNIT;
210 0340 1
211 0341 1 [dsc$sk_dtype_az] :     BEGIN
212 0342 1     BIND chrvec = vms_desc[dsc$a_pointer] : REF VECTOR [,BYTE];
213 0343 1     LOCAL index;
214 0344 1     index = 0;
215 0345 1     WHILE .index LEQ 2046 DO
216 0346 1     BEGIN
```

DBGVALUES
V04-000

N 10
16-Sep-1984 02:45:26 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:17:54 [DEBUG.SRC]DBGVALUES.B32;1

Page 6
(4)

```

: 217 0347 4
: 218 0348 4
: 219 0349 4
: 220 0350 4
: 221 0351 4
: 222 0352 4
: 223 0353 4
: 224 0354 4
: 225 0355 4
: 226 0356 4
: 227 0357 1

```

```

IF .chrvec[.index] EQL 0 THEN EXITLOOP;
index = .index + 1;
END;
length = (.index+1) * %BPUNIT;
END;

[INRANGE,OUTRANGE] : length = .length * %BPUNIT;
TES;

RETURN .length;
END;

! End of 'dbg$data_length'

```

```

.TITLE DBGVALUES
.IDENT \V04-000\

.PSECT DBGSPLIT,NOWRT, SHR, PIC,0

43 41 21 03 00000 P.AAA: .ASCII <3>\!AC\
44 41 21 03 00004 P.AAB: .ASCII <3>\!AD\

.PSECT DBGSOWN,NOEXE, PIC,2

1C 3F 0F C0 00000 SIGNED_DTYPE:
      00 00004 .BYTE -64, 15, 63, 28
      06 00005 .BYTE 0
      .BYTE 6

FORMAT_AC= P.AAA
FORMAT_AD= P.AAB

.EXTRN DBGSGL_CURRENT_PRIMARY
.EXTRN DBGSGB_RADIX, DBGSGB_LANGUAGE
.EXTRN DBGSGL_CONTROL, DBGSGL_CALL_CONTEXT
.EXTRN DBGSGL_CONVERT_TOKEN
.EXTRN DBGSGL_DFLTTP, DBGSGL_DFLTLENG
.EXTRN DBGSGL_SIGN_FLAG
.EXTRN DBGSBUILD_PRIMARY_SUBNODE
.EXTRN DBGSCOLLECT, DBGSCTV_DX_DX
.EXTRN DBGSENUM_POS, DBGSENUM_SUCC
.EXTRN DBGSENUM_VAL, DBGSGET_TEMPMEM
.EXTRN DBGSIS_IT_ENTRY
.EXTRN DBGSINS_DECODE, DBGSLANGUAGE_FORMAT
.EXTRN DBGSNEWLINE, DBGSNGET_RADIX
.EXTRN DBGSPRINT, DBGSPRINT_CONTROL
.EXTRN DBGSPRINT_IDENTIFIER
.EXTRN DBGSPRINT_SET_VALUE
.EXTRN DBGSPRINT_SYMBOL_NAME
.EXTRN DBGSPUSH_TEMPMEM
.EXTRN DBGSPOP_TEMPMEM
.EXTRN DBGSSAVE_VAL, DBGSSTA_SETCONTEXT
.EXTRN DBGSSTA_SYMPATHNAME
.EXTRN DBGSSTA_SYMKIND
.EXTRN DBGSSTA_SYMNAME
.EXTRN DBGSSTA_SYMSIZE
.EXTRN DBGSSTA_SYMTYPE
.EXTRN DBGSSTA_SYMVALUE
.EXTRN DBGSSTA_TYPEFCODE

```


: 0290
: 0306
: 0311
: 0315

[illegible]

DB
VO

PC	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418	Op419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

```
; Routine Size: 181 bytes,   Routine Base: DBG$CODE + 0000
```

```
229 0358 1 GLOBAL ROUTINE DBG$MAKE_SKELETON_DESC(desc_type,data_length) =
230 0359 BEGIN
231 0360 BUILTIN ACTUALCOUNT;
232 0361 LOCAL
233 0362 desc_length,
234 0363 result_desc : REF BLOCK [,LONG] FIELD(dbg$dhdr_fields);
235 0364
236 0365 SELECTONE .desc_type OF
237 0366 SET
238 0367 [dbg$sk_v_value_desc]: desc_length = %UPVAL*dbg$sk_valdesc_base_size + 16;
239 0368
240 0369 [dbg$sk_value_desc]: BEGIN
241 0370 desc_length = %UPVAL*dbg$sk_valdesc_base_size + 16;
242 0371 IF actualcount() GTR 1 THEN
243 0372 IF .data_length GTR 16 THEN
244 0373 desc_length = .data_length + %UPVAL*dbg$sk_valdesc_base_size;
245 0374 END;
246 0375
247 0376 [dbg$sk_primary_desc]: BEGIN
248 0377 desc_length = 20;
249 0378 IF actualcount() GTR 1 THEN
250 0379 desc_length = .desc_length + .data_length;
251 0380 END;
252 0381
253 0382 [OTHERWISE]: SIGNAL();
254 0383 TES;
255 0384
256 0385 result_desc = dbg$get_tempmem((.desc_length + (%UPVAL-1)) / %UPVAL);
257 0386 result_desc[dbg$b_dhdr_lang] = %X'FF';
258 0387 result_desc[dbg$b_dhdr_type] = .desc_type;
259 0388 result_desc[dbg$w_dhdr_length] = .desc_length;
260 0389 RETURN .result_desc;
261 0390
262 0391 END; ! End of 'dbg$make_skeleton_desc'
```

00000083	53	04	AC	D0	00002	.ENTRY	DBG\$MAKE_SKELETON_DESC, Save R2,R3	0358
	8F		53	D1	00006	MOVL	DESC_TYPE, R3	0365
			05	12	0000D	CMPL	R3, #131	0367
	52		30	D0	0000F	BNEQ	18	
			3C	11	00012	MOVL	#48, DESC_LENGTH	
0000007A	8F		53	D1	00014	BRB	48	
			15	12	0001B	CMPL	R3, #122	0369
	52		30	D0	0001D	BNEQ	28	
	01		6C	91	00020	MOVL	#48, DESC_LENGTH	0370
			2B	1B	00023	CMPL	(AP), #1	0371
	10	08	AC	D1	00025	BLEQU	48	
			25	15	00029	CMPL	DATA_LENGTH, #16	0372
52	08	AC	20	C1	0002B	BLEQ	48	
			1E	11	00030	ADDL3	#32, DATA_LENGTH, DESC_LENGTH	0373
00000079	8F		53	D1	00032	BRB	48	0365
			0E	12	00039	CMPL	R3, #121	0376
	52		14	D0	0003B	BNEQ	38	
						MOVL	#20, DESC_LENGTH	0377

DBGVALUES
V04-000

E 11
16-Sep-1984 02:45:26 VAX-11 B11ss-32 V4.0-742
14-Sep-1984 12:17:54 [DEBUG.SRC]DBGVALUES.B32;1

Page 10
(5)

	01		6C	91	0003E	CMPB	(AP), #1	:	0378
			0D	1B	00041	BLEQU	48	:	
	52	08	AC	C0	00043	ADDL2	DATA_LENGTH, DESC_LENGTH	:	0379
			07	11	00047	BRB	48	:	0365
00000000G	00		00	FB	00049	CALLS	#0, LIBSSIGNAL	:	0382
	50		A2	9E	00050	MOVAB	3(R2), R0	:	0385
7E	50	03	04	C7	00054	DIVL3	#4, R0, -(SP)	:	
00000000G	00		01	FB	00058	CALLS	#1, DBG\$GET_TEMPMEM	:	
	03		01	8E	0005F	MNEGB	#1, 3(RESULT_DESC)	:	0386
	02		53	90	00063	MOVB	R3, 2(RESULT_DESC)	:	0387
	60		52	B0	00067	MOVW	DESC_LENGTH, -(RESULT_DESC)	:	0388
				04	0006A	RET		:	0391

; Routine Size: 107 bytes, Routine Base: DBG\$CODE + 00B5

```

264 0392 1 GLOBAL ROUTINE DBG$MAKE_INTEGER_DESC(VALUE) =
265 0393 1
266 0394 1 ROUTINE DESCRIPTION
267 0395 1     Given an integer value, this routine builds a value descriptor
268 0396 1     for that integer.
269 0397 1
270 0398 1 INPUTS
271 0399 1     VALUE    - The integer value
272 0400 1
273 0401 1 OUTPUTS
274 0402 1     A pointer to the constructed value descriptor is returned.
275 0403 1     The descriptor is built out of temporary memory.
276 0404 1
277 0405 1 BEGIN
278 0406 1 LOCAL
279 0407 1     TEMP_DESC: REF DBG$VALDESC;
280 0408 1
281 0409 1     TEMP_DESC = DBG$MAKE_SKELETON_DESC(DBG$SK_VALUE_DESC);
282 0410 1     TEMP_DESC[DBG$B_DHDR_KIND] = RST$K_DATA;
283 0411 1     TEMP_DESC[DBG$B_DHDR_FCODE] = RST$K_TYPE_ATOMIC;
284 0412 1     TEMP_DESC[DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
285 0413 1     TEMP_DESC[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_L;
286 0414 1     TEMP_DESC[DBG$W_VALUE_LENGTH] = 4;
287 0415 1     TEMP_DESC[DBG$L_VALUE_POINTER] = TEMP_DESC[DBG$A_VALUE_ADDRESS];
288 0416 1     TEMP_DESC[DBG$L_VALUE_VALUE0] = .VALUE;
289 0417 1 RETURN .TEMP_DESC;
290 0418 1 END;

```

```

      7E      7A      0000 0000
8B AF      8F 9A 00002
06 A0      01 FB 00006
14 A0 0602 8F B0 0000A
18 A0 01080004 8F D0 00010
20 A0      20 A0 9E 00018
      04 AC D0 0001D
      04 00022

```

```

.ENTRY DBG$MAKE_INTEGER_DESC, Save nothing
MOVZBL #122, -(SP)
CALLS #1, DBG$MAKE_SKELETON_DESC
MOVW #1538, 6(TEMP_DESC)
MOVL #17301508, 20(TEMP_DESC)
MOVAB 32(TEMP_DESC), 24(TEMP_DESC)
MOVL VALUE, 32(TEMP_DESC)
RET

```

```

: 0392
: 0409
:
: 0411
: 0414
: 0415
: 0416
: 0418

```

; Routine Size: 35 bytes. Routine Base: DBG\$CODE + 0120

```
292 0419 1 GLOBAL ROUTINE DBG$MAKE_VAL_DESC (desc_ptr,target_type) =
293 0420 1
294 0421 1 ROUTINE DESCRIPTION
295 0422 1 Given a VMS descriptor, this routine builds either a Value Descriptor
296 0423 1 or a Volatile Value Descriptor around the VMS descriptor.
297 0424 1 In the case where a Value Descriptor is constructed, we need to
298 0425 1 extract the value represented by the VMS descriptor, and put
299 0426 1 this value inside the Value Descriptor. So, for descriptors
300 0427 1 representing bitfields, this routine is where the actual extraction
301 0428 1 of the bits takes place.
302 0429 1
303 0430 1 INPUTS
304 0431 1 DESC_PTR - Points to a Vax-standard VMS descriptor
305 0432 1 TARGET_TYPE - A constant which can be one of:
306 0433 1 DBG$K_VALUE_DESC or DBG$K_V_VALUE_DESC
307 0434 1
308 0435 1 OUTPUTS
309 0436 1 A Value Descriptor or a Volatile Value Descriptor is constructed
310 0437 1 out of temporary memory. A pointer to this descriptor is returned.
311 0438 1
312 0439 1 BEGIN
313 0440 1 LOCAL
314 0441 1 vms_desc : dbg$stg_desc,
315 0442 1 bits,bytes,
316 0443 1 result_desc : REF dbg$valdesc;
317 0444 1
318 0445 1 * The first thing we do is to 'de-reference' data items of type
319 0446 1 * descriptor which is what we get for arrays of dynamic strings.
320 0447 1 *
321 0448 1 ch$move(12,.desc_ptr,vms_desc);
322 0449 1 IF .target_type EQL dbg$K_value_desc THEN
323 0450 1 WHILE .vms_desc[dsc$b_dtype] EQL dsc$K_dtype_desc DO
324 0451 1 BEGIN
325 0452 1 BUILTIN PROBER;
326 0453 1 LOCAL addr;
327 0454 1 addr = .vms_desc[dsc$a_pointer];
328 0455 1 IF NOT PROBER(%REF(0),%REF(8),.addr) THEN SIGNAL(dbg$noaccessr,1,.addr);
329 0456 1 ch$move(8,.addr,vms_desc);
330 0457 1 CASE .vms_desc[dsc$b_class] FROM dsc$K_class_2 TO dsc$K_class_ubs
331 0458 1 OF SET
332 0459 1 [dsc$K_class_s,dsc$K_class_d,dsc$K_class_vs] :
333 0460 1 BEGIN
334 0461 1 IF .vms_desc[dsc$b_class] EQL dsc$K_class_d
335 0462 1 THEN vms_desc[dsc$b_class] = dsc$K_class_s;
336 0463 1 IF .vms_desc[dsc$b_dtype] EQL dsc$K_dtype_vt
337 0464 1 THEN vms_desc[dsc$b_class] = dsc$K_class_vs;
338 0465 1 IF .vms_desc[dsc$b_class] EQL dsc$K_class_vs
339 0466 1 AND .vms_desc[dsc$b_dtype] EQL dsc$K_dtype_t
340 0467 1 THEN vms_desc[dsc$b_dtype] = dsc$K_dtype_vt;
341 0468 1 vms_desc[dsc$l_pos] = 0;
342 0469 1 END;
343 0470 1
344 0471 1 [dsc$K_class_sd,dsc$K_class_ubs] :
345 0472 1 BEGIN
346 0473 1 IF NOT PROBER(%REF(0),%REF(4),.addr+8) THEN SIGNAL(dbg$noaccessr,1,.addr+8);
347 0474 1 vms_desc[dsc$l_pos] = .(.addr+8)<0,32,0>;
348 0475 1 END;
```



```
[INRANGE,OUTRANGE] :      SIGNAL(dbg$_illtype);
TES;
END;
+
+ Obtain the length in bits and the length in bytes of the data
+ represented by the VMS descriptor.
-
bits = dbg$_data_length(vms_desc);
bytes = (.bits + (%BPUNIT-1)) / %BPUNIT;
+
+ Allocate either enough space for a Volatile Value Descriptor,
+ or enough space for a Value Descriptor which contains the value.
-
IF (.target_type EQL dbg$_k_v_value_desc) OR (.bytes GTR 512)
THEN
    result_desc = dbg$_make_skeleton_desc(dbg$_k_v_value_desc)
ELSE
    result_desc = dbg$_make_skeleton_desc(dbg$_k_value_desc,.bytes);
+
+ Copy the VMS descriptor into the Value Descriptor. Also fill in
+ the kind and fcode fields.
-
ch$move(12,vms_desc,result_desc[dbg$_a_value_vmsdesc]);
result_desc[dbg$_b_dhdr_kind] = rst$_k_data;
result_desc[dbg$_b_dhdr_fcode] = rst$_k_type_descr;
+
+ If the target type is a value descriptor then we want to extract
+ the value represented by the VMS descriptor and
+ right-align the result in the value descriptor.
-
IF .result_desc[dbg$_b_dhdr_type] EQL dbg$_k_value_desc THEN
    BEGIN
        BUILTIN PROBER;
        LOCAL addr,pos;
        result_desc[dbg$_l_value_pointer] = result_desc[dbg$_a_value_address];
        +
        + Compute the byte address by adding (bit_offset/8).
        + Then set the bit offset to (bit_offset mod 8).
        -
        IF .vms_desc[dsc$_b_class] EQL dsc$_k_class_ubs
        THEN
            BEGIN
                pos = .(vms_desc[dsc$_l_pos])<0,3,0>;
                addr = .vms_desc[dsc$_a_pointer] + .(vms_desc[dsc$_l_pos])<3,29,1>;
            END
        ELSE
            BEGIN
                pos = 0;
                addr = .vms_desc[dsc$_a_pointer];
            END;
        +
        + Check for read access.
        -
```

```
bytes = (.pos + .bits + 7)/8;
IF .bytes NEQ 0
THEN
  IF NOT PROBER(%REF(0),bytes,..addr)
  THEN
    SIGNAL(dbg$_noaccessr,1,..addr);
  +
  Don't support bit extractions bigger than 32 bits for now.
  This restriction may be relaxed at a later time.
  -
  IF .bits LEQU 32
  THEN
    BEGIN
      +
      Decide whether to do a signed or an unsigned extraction based on
      the dtype.
      -
      SELECTONE .vms_desc[dsc$b_dtype] OF
      SET
        [dsc$b_dtype_sv,dsc$b_dtype_svu,dsc$b_dtype_b,dsc$b_dtype_w] :
        .result_desc[dbg$_value_pointer] = .(.addr)<.pos,.bits,1>;
      [OTHERWISE] :
        .result_desc[dbg$_value_pointer] = .(.addr)<.pos,.bits,0>;
      TES;
      +
      Since we have performed the bit extraction, the bit offset is
      now zero. Zero the POS field to reflect this.
      -
      IF .result_desc[dbg$b_value_class] EQL dsc$b_class_ubs
      THEN result_desc[dbg$_value_pos] = 0;
    END
  ELSE
    BEGIN
      +
      The value is longer than 32 bits.
      -
      IF .pos EQL 0
      THEN
        +
        Copy the bytes.
        -
        ch$move(.bytes,..addr,.result_desc[dbg$_value_pointer])
      ELSE
        BEGIN
          +
          We used to disallow this.
          SIGNAL(dbg$_unimplent);
          -
          INCR i FROM 0 TO (.bits-1)/32 DO
            (4*.i + .result_desc[dbg$_value_pointer]) =
              (.addr)<.pos,32,0>;
          +
          Since we have performed the bit extraction, the bit offset is
          now zero. Zero the POS field to reflect this.
```

```

463      0590      S
464      0591      S
465      0592      S
466      0593      S
467      0594      S
468      0595      S
469      0596      S
470      0597      S
471      0598      S
472      0599      S
473      0600      S
474      0601      S
475      0602      S

      IF .result_desc[dbg$b_value_class] EQL dsc$h_class_ubs
      THEN result_desc[dbg$l_value_pos] = 0;
      END;
      END;
      END;

      !+
      !- We are all done. Return a pointer to the newly constructed value
      !- Descriptor.
      RETURN .result_desc;
      END; ! end of routine dbg$make_val_desc
```

```

                                OFFC 00000
                                .ENTRY DBG$MAKE VAL DESC, Save R2,R3,R4,R5,R6,R7,-
                                R8,R9,R10,R11
                                0419
                                6E      04      SE      0C      C2      00002
                                0000007A      BC      0C      28      00005
                                8F      08      AC      D1      0000A
                                18      02      04      12      00012
                                0098      31      0001A
                                56      04      AE      D0      0001D
                                08      00      0C      00021
                                11      12      00025
                                56      DD      00027
                                00028228      01      DD      00029
                                00000000G      00      8F      DD      0002B
                                6E      00      03      FB      00031
                                0D      66      08      28      00038
                                001C      002B      002B      001C      00041
                                001C      001C      001C      00049
                                002B      001C      0054      00051
                                0054      0054      001C      00059
                                00000000G      00      8F      DD      0005D
                                01      FB      00063
                                02      03      AB      11      0006A
                                04      12      AE      91      0006C
                                03      AE      01      90      00072
                                25      02      AE      91      00076
                                68:
                                75:
                                85:
                                95:
                                125:
                                165:
                                165848
                                0448
                                0449
                                0450
                                0454
                                0455
                                0456
                                0457
                                0477
                                0461
                                0462
                                0463
```


			03	AE		04	12	0007A	BNEQ	108				
				0B		0B	90	0007C	MOVB	#11, VMS_DESC+3	0464			
					03	AE	91	00080	108:	CMPB	VMS_DESC+3, #11	0465		
				0E		0A	12	00084	BNEQ	118				
					02	AE	91	00086	CMPB	VMS_DESC+2, #14	0466			
						04	12	0008A	BNEQ	118				
			02	AE		25	90	0008C	MOVB	#37, VMS_DESC+2	0467			
					08	AE	D4	00090	118:	CLRL	VMS_DESC+8	0468		
						D5	11	00093	BRB	78	0457			
08	A6			04		00	0C	00095	128:	PROBER	#0, #4, 8(ADDR)	0473		
						12	12	0009A	BNEQ	138				
					08	A6	9F	0009C	PUSHAB	8(ADDR)				
						01	DD	0009F	PUSHL	#1				
						8F	DD	000A1	PUSHL	#164392				
		00000000G		00	00028228	03	FB	000A7	CALLS	#3, LIB\$SIGNAL				
			08	AE		08	A6	D0	000AE	138:	MOVL	8(ADDR), VMS_DESC+8	0474	
						B5	11	000B3	BRB	78	0450			
						5E	DD	000B5	148:	PUSHL	SP	0484		
		FE01		CF		01	FB	000B7	CALLS	#1, DBG\$DATA_LENGTH				
				5A		50	D0	000BC	MOVL	R0, BITS				
				50		07	AA	9E	000BF	MOVAB	7(R10), R0	0485		
			5B	50		08	C7	000C3	DIVL3	#8, R0, BYTES				
		00000083		8F		08	AC	D1	000C7	CMPL	TARGET_TYPE, #131	0491		
						09	13	000CF	BEQL	158				
		00000200		8F		5B	D1	000D1	CMPL	BYTES, #512				
						0B	15	000D8	BLEQ	168				
				7E		83	8F	9A	000DA	158:	MOVZBL	#131, -(SP)	0493	
		FE8F		CF		01	FB	000DE	CALLS	#1, DBG\$MAKE_SKELETON_DESC				
						0B	11	000E3	BRB	178				
						5B	DD	000E5	168:	PUSHL	BYTES	0495		
				7E		7A	8F	9A	000E7	MOVZBL	#122, -(SP)			
		FE82		CF		02	FB	000EB	CALLS	#2, DBG\$MAKE_SKELETON_DESC				
				57		50	D0	000F0	178:	MOVL	R0, RESULT_DESC			
				6E		0C	28	000F3	MOVCL3	#12, VMS_DESC, 20(RESULT_DESC)	0501			
14	A7			A7	0603	8F	B0	000F8	MOVW	#1539, 6(RESULT_DESC)	0503			
				7A	02	A7	91	000FE	CMPB	2(RESULT_DESC), #122	0510			
						7E	12	00103	BNEQ	258				
				56		18	A7	9E	00105	MOVAB	24(RESULT_DESC), R6	0514		
				66		20	A7	9E	00109	MOVAB	32(R7), (R6)			
				0D		03	AE	91	0010D	CMPB	VMS_DESC+3, #13	0519		
						12	12	00111	BNEQ	188				
58	08	AE		03		00	EF	00113	EXTZV	#0, #3, VMS_DESC+8, POS	0522			
		59		AE		8F	78	00119	ASHL	#-3, VMS_DESC+8, ADDR	0523			
			08	59		04	AE	C0	0011F	ADDL2	VMS_DESC+4, ADDR			
						06	11	00123	BRB	198	0519			
						5B	D4	00125	188:	CLRL	POS	0527		
				59		04	AE	D0	00127	MOVL	VMS_DESC+4, ADDR	0528		
				50		07	AA	48	9E	0012B	198:	MOVAB	7(BITS)[POS], R0	0533
				50		08	C7	00130	DIVL3	#8, R0, BYTES				
		5B				17	13	00134	BEQL	208	0534			
						00	0C	00136	PROBER	#0, BYTES, (ADDR)	0536			
		69		5B		11	12	0013A	BNEQ	208				
						59	DD	0013C	PUSHL	ADDR	0538			
						01	DD	0013E	PUSHL	#1				
						8F	DD	00140	PUSHL	#164392				
		00000000G		00	00028228	03	FB	00146	CALLS	#3, LIB\$SIGNAL				
				20		5A	D1	0014D	208:	CMPL	BITS, #32	0543		

			50	02	28	1A	00150	BGTRU	248			
			06		AE	9A	00152	MOVZBL	VMS_DESC+2, R0			0550
					50	91	00156	CMPB	R0, #6			0552
					05	1F	00159	BLSSU	218			
			07		50	91	0015B	CMPB	R0, #7			
					0A	1B	0015E	BLEQU	228			
			29		50	91	00160	CMPB	R0, #41			
					0D	1F	00163	BLSSU	238			
			2A		50	91	00165	CMPB	R0, #42			
					08	1A	00168	BGTRU	238			
00	B6	69	5A		58	EE	0016A	EXTV	POS, BITS, (ADDR), 20(R6)			0553
					2D	11	00170	BRB	298			
00	B6	69	5A		58	EF	00172	EXTZV	POS, BITS, (ADDR), 20(R6)			0555
					25	11	00178	BRB	298			0561
					58	D5	0017A	TSTL	POS			0570
					07	12	0017C	BNEQ	268			
		00	B6	69	58	28	0017E	MOV3	BYTES, (ADDR), 20(R6)			0575
					23	11	00183	BRB	308			
			50	FF	AA	9E	00185	MOVAB	-1(R10), R0			0583
			50		20	C6	00189	DIVL2	#32, R0			
			51		01	CE	0018C	MNEGL	#1, I			
					0A	11	0018F	BRB	288			
					69	41	DF	00191	PUSHAL	(ADDR)[I]		0585
00	B641	9E	20		58	EF	00194	EXTZV	POS, #32, 2(SP)+, 20(R6)[I]			
		F2	51		50	F3	0019B	AOBLEQ	R0, I, 278			0584
			0D	17	A7	91	0019F	CMPB	23(RESULT_DESC), #13			0591
					03	12	001A3	BNEQ	308			
				1C	A7	D4	001A5	CLRL	28(RESULT_DESC)			0592
			50		57	D0	001AB	MOVL	RESULT_DESC, R0			0601
					04	001AB		RET				0602

; Routine Size: 428 bytes, Routine Base: DBG\$CODE + 0143

```
477 0603 1 GLOBAL ROUTINE DBG$FILL_IN_VMS_DESC(fcode,typeid,symid,  
478 0604                                vms_desc,bit_length,bit_offset) =  
479 0605  
480 0606  
481 0607  
482 0608  
483 0609  
484 0610  
485 0611  
486 0612  
487 0613  
488 0614  
489 0615  
490 0616  
491 0617  
492 0618  
493 0619  
494 0620  
495 0621  
496 0622  
497 0623  
498 0624  
499 0625  
500 0626  
501 0627  
502 0628  
503 0629  
504 0630  
505 0631  
506 0632  
507 0633  
508 0634  
509 0635  
510 0636  
511 0637  
512 0638  
513 0639  
514 0640  
515 0641  
516 0642  
517 0643  
518 0644  
519 0645  
520 0646  
521 0647  
522 0648  
523 0649  
524 0650  
525 0651  
526 0652  
527 0653  
528 0654  
529 0655  
530 0656  
531 0657  
532 0658  
533 0659
```

```
ROUTINE DESCRIPTION  
BEGIN  
MAP  
    symid      : REF rst$entry,  
    vms_desc   : REF dbg$stg_desc,  
    bit_length : REF VECTOR [1, LONG],  
    bit_offset : REF VECTOR [1, LONG];  
CASE fcode FROM rst$sk_type_minimum TO rst$sk_type_maximum OF  
SET  
[rst$sk_type_atomic] :  
    BEGIN  
    LOCAL typecode;  
    + Atomic data types - the routine dbg$sta_typ_atomic can be  
    used to obtain the dtype and length in bits. Class is set  
    to S or VS here; it may later be changed to UBS if there  
    is a bit offset present.  
    -  
    dbg$sta_typ_atomic(.typeid, typecode, bit_length[0]);  
    IF .typecode EQL dsc$sk_bool THEN  
    BEGIN  
        vms_desc[dsc$b_class] = dsc$sk_class_s;  
        vms_desc[dsc$b_dtype] = dsc$sk_dtype_tf;  
        vms_desc[dsc$b_length] = .bit_length[0];  
    END  
    ELSE  
    BEGIN  
        vms_desc[dsc$b_dtype] = .typecode;  
        IF .typecode EQL dsc$sk_dtype_vt  
        THEN  
            vms_desc[dsc$b_class] = dsc$sk_class_vs  
        ELSE  
            vms_desc[dsc$b_class] = dsc$sk_class_s;  
        + Length is in bits for the five data types below, and  
        bytes for all others.  
        -  
        vms_desc[dsc$b_length] = .bit_length[0]/  
        (IF .typecode EQL dsc$sk_dtype_v  
        OR .typecode EQL dsc$sk_dtype_vu  
        OR .typecode EQL dsc$sk_dtype_sv  
        OR .typecode EQL dsc$sk_dtype_svu  
        OR .typecode EQL dsc$sk_dtype_tf  
        THEN 1  
        ELSE %BPUNIT);  
    END;  
END;  
[rst$sk_type_pict] :  
    BEGIN  
    +
```



```

! "Pictured" data item. This is really an item of data-type
! dsc$dtype t, but we need to retain extra information in
! case we want to deposit a numeric value into this item.
-
dbg$sta_symsize (.typeid, bit_length[0]);
vms_desc[dsc$b_class] = dsc$dtype_t;
vms_desc[dsc$b_dtype] = dsc$dtype_t;
vms_desc[dsc$b_length] = .bit_length[0]/%BPUNIT;
END;

[rst$dtype_record, rst$dtype_ptr, rst$dtype_tptr,
rst$dtype_enum, rst$dtype_set, rst$dtype_subrng,
rst$dtype_file, rst$dtype_rfa]:
-
! Non-atomic (i.e., non-VAX standard) data types. For these,
! we do not attempt to fill in the VMS descriptor. We just
! fill in the bit_length.
-
dbg$sta_symsize(.typeid, bit_length[0]);

! For variants, there is nothing to fill in.
[rst$dtype_variant]:
0;

[rst$dtype_descr] :
BEGIN
-
! Types described by descriptors. We use the routine dbg$sta_typ_descr
! to obtain the class, dtype, and length information.
-
LOCAL dst_desc : REF dbg$stg_desc;
dbg$sta_typ_descr(.typeid, dst_desc);

! If we got a symid passed in to this routine,
! try calling SYMVALUE to get a
! descriptor. If we get one, then use this
! descriptor instead of the one we got back from STA_TYP_DESCR.

! Note - normally, these 2 descriptors will be the same.
! However, for dynamic arrays in PASCAL, the runtime descriptor
! (which we get back when we call SYMVALUE with the symid) is
! correct, but the compile-time descriptor (which is part of
! the typespec) is wrong. This code is a workaround for this
! problem in the PASCAL DST. The same workaround appears
! in DBGPARSER for array descriptors.

IF .SYMID NEQ 0
THEN
BEGIN
LOCAL
DESC: VECTOR[3],
RSTPTR: REF RST$ENTRY,
VALUE_KIND;
RSTPTR = .SYMID;

```

```
591      WHILE .RSTPTR[RST$B_KIND] NEQ RST$K_MODULE DO
592          RSTPTR = .RSTPTR[RST$L_UPSCOPEPTR];
593      IF .RSTPTR[RST$B_LANGUAGE] EQL DBG$K_PASCAL
594      THEN
595          BEGIN
596              DBG$STA_SETCONTEXT(.SYMID);
597              DBG$STA_SYMVALUE(.SYMID, DESC, VALUE_KIND);
598              IF .VALUE_KIND EQL DBG$K_VAL_DESCR
599              THEN
600                  DST_DESC = .DESC[0];
601              END;
602          END;
603      END;
604      vms_desc[dsc$b_class] = .dst_desc[dsc$b_class];
605      vms_desc[dsc$b_dtype] = .dst_desc[dsc$b_dtype];
606      vms_desc[dsc$b_length] = .dst_desc[dsc$b_length];
607
608      !+
609      ! Fix things up so that dtype VT always corresponds to class VS.
610      ! (This seems to be necessary for PL/I varying strings).
611      !-
612      IF .vms_desc[dsc$b_dtype] EQL dsc$K_dtype_vt
613      THEN
614          vms_desc[dsc$b_class] = dsc$K_class_vs;
615
616      SELECTONE .vms_desc[dsc$b_class] OF
617          SET
618          [dsc$K_class_s, dsc$K_class_d, dsc$K_class_vs] : 0;
619
620          [dsc$K_class_sd] :
621              BEGIN
622                  vms_desc[dsc$b_digits] = .dst_desc[dsc$b_digits];
623                  vms_desc[dsc$b_scale] = .dst_desc[dsc$b_scale];
624                  vms_desc[dsc$b_fl_binscale] = .dst_desc[dsc$b_fl_binscale];
625
626                  !+
627                  ! *** Workaround for a problem in the PL/I DST.
628                  ! *** The scale they are giving us is the negative
629                  ! *** of what we expect.
630                  !-
631                  IF .symid NEQ 0
632                  THEN
633                      BEGIN
634                          WHILE .symid[rst$b_kind] NEQ rst$K_module DO
635                              symid = .symid[rst$l_upscopeptr];
636                          IF (.symid[rst$b_language] EQL dbg$K_pli) AND
637                          .symid[rst$b_oldpliflag]
638                          THEN
639                              vms_desc[dsc$b_scale] = - .dst_desc[dsc$b_scale];
640                          END;
641                      END;
642
643          [dsc$K_class_ubs] :
644              SELECTONE .dst_desc[dsc$b_dtype] OF
645                  SET
646                  [dsc$K_dtype_sy, dsc$K_dtype_vu, dsc$K_dtype_tf]:
647                      bit_offset[0] = .bit_offset[0] + .dst_desc[dsc$l_pos];
```

```

648 0774
649 0775
650 0776
651 0777
652 0778
653 0779
654 0780
655 0781
656 0782
657 0783
658 0784
659 0785
660 0786
661 0787
662 0788
663 0789
664 0790
665 0791
666 0792
667 0793
668 0794
669 0795
670 0796
671 0797
672 0798
673 0799
674 0800
675 0801
676 0802
677 0803
678 0804
679 0805
680 0806
681 0807
682 0808
683 0809
684 0810
685 0811
686 0812
687 0813
688 0814
689 0815
690 0816
691 0817
692 0818
693 0819
694 0820
695 0821
696 0822
697 0823
698 0824
699 0825
700 0826
701 0827
702 0828
703 0829
704 0830

[dsc$k_dtype_ubs]:
BEGIN
  bit_offset[0] = bit_offset[0] + (.dst_desc+8)<0,16,1>;
  IF (.dst_desc+10)<0,1,0>
    THEN vms_desc[dsc$b_dtype] = dsc$k_dtype_svu
    ELSE vms_desc[dsc$b_dtype] = dsc$k_dtype_vu;
  END;
[OTHERWISE]:
  0;
  TES;
[OTHERWISE] :
  SIGNAL(dbg$_unimplent);
  TES;
IF .vms_desc[dsc$b_dtype] EQL dsc$k_dtype_bpv
THEN
  BEGIN
    vms_desc[dsc$b_class] = dsc$k_class_z;
    vms_desc[dsc$b_dtype] = dsc$k_dtype_zem;
    vms_desc[dsc$w_length] = 2;
    dbg$gl_call_context = .dst_desc[dsc$a_frame];
  END
ELSE IF .vms_desc[dsc$b_dtype] EQL dsc$k_dtype_blv
THEN
  BEGIN
    vms_desc[dsc$b_class] = dsc$k_class_z;
    vms_desc[dsc$b_dtype] = dsc$k_dtype_zi;
    vms_desc[dsc$w_length] =
      (dbg$ins_decode(.vms_desc[dsc$a_pointer],false,false) -
       .vms_desc[dsc$a_pointer]);
    dbg$gl_call_context = .dst_desc[dsc$a_frame];
  END;

  bit_length[0] = dbg$data_length(.vms_desc);
  END;

! Self relative labels in PL/I (i.e., arrays of labels).
! The value of one of these is equal to the contents of the
! memory location plus its own address. In other words, the
! values actually stored in the label array are offsets to
! the actual place to branch to.
[rst$k_type_self_rel_lab]:
BEGIN
  vms_desc[dsc$b_class] = dsc$k_class_z;
  vms_desc[dsc$b_dtype] = dsc$k_dtype_zi;
  vms_desc[dsc$a_pointer] = .vms_desc[dsc$a_pointer] +
    (.vms_desc[dsc$a_pointer]);
  vms_desc[dsc$w_length] =
    (dbg$ins_decode(.vms_desc[dsc$a_pointer],false,false) -
     .vms_desc[dsc$a_pointer]);
  bit_length[0] = .vms_desc[dsc$w_length] * 8;
  END;
```



```
.. 705      0831      2      ! We do not handle any other fcodes.
.. 706      0832      2
.. 707      0833      2
.. 708      0834      2      [INRANGE,OUTRANGE] :
.. 709      0835      2      SIGNAL(dbg$_unimplent);
.. 710      0836      2      TES;
.. 711      0837      2      RETURN sts$_success;
.. 712      0838      1      END;                                ! End of routine 'dbg$fill_in_vms_desc'
```

				03FC 00000	.ENTRY	DBG\$FILL_IN_VMS_DESC, Save R2,R3,R4,R5,R6,-			
			59	00000000G	00 9E	00002	MOVAB	LIB\$SIGNAL, R9	
			58	00000000G	00 9E	00009	MOVAB	DBG\$INS_DECODE, R8	
			57	00000000G	00 9E	00010	MOVAB	DBG\$STA_SYMSIZE, R7	
			5E		18 C2	00017	SUBL2	#24, SP	
			01	04	AC CF	0001A	CASEL	FCODE, #1, #21	0603
00C6	00D2		0037		002C	0001F	.WORD	28-18,-	0615
00C6	00C6		00C6		00A9	00027		38-18,-	
002C	002C		002C		00C6	0002F		148-18,-	
00C6	00C6		002C		002C	00037		128-18,-	
00C6	0234		002C		002C	0003F		118-18,-	
			002C		0212	00047		128-18,-	
								128-18,-	
								128-18,-	
								128-18,-	
								28-18,-	
								28-18,-	
								28-18,-	
								28-18,-	
								28-18,-	
								28-18,-	
								128-18,-	
								128-18,-	
								28-18,-	
								28-18,-	
								338-18,-	
								128-18,-	
								328-18,-	
								28-18	
			00028800	8F	DD	0004B	PUSHL	#165888	0834
		69		01	FB	00051	CALLS	#1, LIB\$SIGNAL	
				70	11	00054	BRB	108	
			14	AC	DD	00056	PUSHL	BIT_LENGTH	0626
			04	AE	9F	00059	PUSHAB	TYPECODE	
			08	AC	DD	0005C	PUSHL	TYPEID	
00000000G	00			03	FB	0005F	CALLS	#3, DBG\$STA_TYP_ATOMIC	
	50		10	AC	DD	00066	MOVL	VMS_DESC, R0	0629
	51		10	AC	DD	0006A	MOVL	VMS_DESC, R1	0630
	52			6E	DD	0006E	MOVL	TYPECODE, R2	0627
0000009E	8F			52	D1	00071	CMPL	R2, #158	
				0F	12	00078	BNEQ	48	
03	A0			01	90	0007A	MOVB	#1, 3(R0)	0629
02	A1			28	90	0007E	MOVB	#40, 2(R1)	0630
10	BC		14	BC	B0	00082	MOVW	@Bif_LENGTH, @VMS_DESC	0631

	02	A1		65	11	00087	BRB	13\$	0627	
		25		52	90	00089	4\$: MOVB	R2, 2(R1)	0635	
				52	D1	0008D	CMPL	R2, #37	0636	
	03	A0		06	12	00090	BNEQ	5\$		
				08	90	00092	MOVB	#11, 3(R0)	0638	
	03	A0		04	11	00096	BRB	6\$		
		01		01	90	00098	5\$: MOVB	#1, 3(R0)	0640	
				52	D1	0009C	6\$: CMPL	R2, #1	0646	
		22		14	13	0009F	BEQL	7\$		
				52	D1	000A1	CMPL	R2, #34	0647	
		29		0F	13	000A4	BEQL	7\$		
				52	D1	000A6	CMPL	R2, #41	0648	
		2A		0A	13	000A9	BEQL	7\$		
				52	D1	000AB	CMPL	R2, #42	0649	
		28		05	13	000AE	BEQL	7\$		
				52	D1	000B0	CMPL	R2, #40	0650	
		50		05	12	000B3	BNEQ	8\$		
				01	D0	000B5	7\$: MOVL	#1, R0	0646	
		50		03	11	000B8	BRB	9\$		
51				08	D0	000BA	8\$: MOVL	#8, R0		
	14	BC		50	C7	000BD	9\$: DIVL3	R0, @BIT_LENGTH, R1		
	10	BC		51	B0	000C2	MOVW	R1, @VMS_DESC		
				26	11	000C6	10\$: BRB	13\$	0615	
			14	AC	DD	000C8	11\$: PUSHL	BIT_LENGTH	0664	
			08	AC	DD	000CB	PUSHL	TYPEID		
		67		02	FB	000CE	CALLS	#2, DBG\$STA_SYMSIZE		
		50		10	AC	DD	000D1	MOVL	VMS_DESC, R0	0665
51	02	A0	010E	8F	B0	000D5	MOVW	#270, 2(R0)	0666	
	14	BC		08	C7	000DB	DIVL3	#8, @BIT_LENGTH, R1	0667	
		60		51	B0	000E0	MOVW	R1, (R0)		
				09	11	000E3	BRB	13\$	0615	
			14	AC	DD	000E5	12\$: PUSHL	BIT_LENGTH	0679	
			08	AC	DD	000E8	PUSHL	TYPEID		
		67		02	FB	000EB	CALLS	#2, DBG\$STA_SYMSIZE		
			0162	31	000EE	13\$: BRW	33\$			
			04	AE	9F	000F1	14\$: PUSHAB	DST_DESC	0694	
			08	AC	DD	000F4	PUSHL	TYPEID		
00000000G	00			02	FB	000F7	CALLS	#2, DBG\$STA_TYP_DESCR		
	52		0C	AC	D0	000FE	MOVL	SYMD, R2	0709	
				56	D4	00102	CLRL	R6		
				52	D5	00104	TSTL	R2		
				3A	13	00106	BEQL	17\$		
				56	D6	00108	INCL	R6		
	50			52	D0	0010A	MOVL	R2, RSTPTR	0716	
	01		14	A0	91	0010D	15\$: CMPB	20(RSTPTR), #1	0717	
				06	13	00111	BEQL	16\$		
	50		10	A0	D0	00113	MOVL	16(RSTPTR), RSTPTR	0718	
				F4	11	00117	BRB	15\$		
	06		29	A0	91	00119	16\$: CMPB	41(RSTPTR), #6	0719	
				23	12	0011D	BNEQ	17\$		
				52	DD	0011F	PUSHL	R2	0722	
00000000G	00			01	FB	00121	CALLS	#1, DBG\$STA_SETCONTEXT		
			08	AE	9F	00128	PUSHAB	VALUE_KIND	0723	
			10	AE	9F	0012B	PUSHAB	DESC		
				52	DD	0012E	PUSHL	R2		
00000000G	00			03	FB	00130	CALLS	#3, DBG\$STA_SYMVALUE		
	03		08	AE	D1	00137	CMPL	VALUE_KIND, -#3	0724	

04	AE	OC	05	12	00138	BNEQ	178			
	52	10	AE	00	0013D	MOVL	DESC, DST_DESC			0726
	54	03	AC	00	00142	178:	MOVL	VMS_DESC, R2		0730
	53	04	A2	9E	00146	MOVAB	3(R2), R4			
	64	03	AE	00	0014A	MOVL	DST_DESC, R3			
	55	02	A3	90	0014E	MOVB	3(R3), (R4)			
	50	02	A2	9E	00152	MOVAB	2(R2), R5			0731
	65		A3	9A	00156	MOVZBL	2(R3), R0			
	62		50	90	0015A	MOVB	R0, (R5)			
	25		63	80	0015D	MOVW	(R3), (R2)			0732
			65	91	00160	CMPB	(R5), #37			0738
			03	12	00163	BNEQ	188			
	64		08	90	00165	MOVB	#11, (R4)			0740
			64	95	00168	188:	TSTB	(R4)		0744
			05	13	0016A	BEQL	198			
	02		64	91	0016C	CMPB	(R4), #2			
			7A	18	0016F	BLEQU	268			
	08		64	91	00171	198:	CMPB	(R4), #11		
			75	13	00174	BEQL	268			
	09		64	91	00176	CMPB	(R4), #9			0746
			38	12	00179	BNEQ	228			
0A	50	0A	A3	80	0017B	MOVW	8(R3), 8(R2)			0749
A2	01		03	EF	00180	EXTZV	#3, #1, 10(R3), R0			0750
	03		50	F0	00186	INSV	R0, #3, #1, 10(R2)			
	67		56	E9	0018C	BLBC	R6, 288			0757
	50	OC	AC	00	0018F	208:	MOVL	SYMID, R0		0760
	01	14	A0	91	00193	CMPB	20(R0), #1			
			07	13	00197	BEQL	218			
OC	AC	10	A0	00	00199	MOVL	16(R0), SYMID			0761
	50	OC	EF	11	0019E	BRB	208			
	05	29	AC	00	001A0	218:	MOVL	SYMID, R0		0762
			A0	91	001A4	CMPB	41(R0), #5			
			4C	12	001A8	BNEQ	288			
47	28		05	E1	001AA	BBC	#5, 40(R0), 288			0763
	08	08	A3	8E	001AF	MNEGB	8(R3), 8(R2)			0765
			40	11	001B4	BRB	288			0742
	0D		64	91	001B6	228:	CMPB	(R4), #13		0769
			32	12	001B9	BNEQ	278			
	22		50	91	001BB	CMPB	R0, #34			0772
			0A	13	001BE	BEQL	238			
	28		50	91	001C0	CMPB	R0, #40			
			05	13	001C3	BEQL	238			
	2A		50	91	001C5	CMPB	R0, #42			
			07	12	001C8	BNEQ	248			
18	BC	08	A3	C0	001CA	238:	ADDL2	8(R3), @BIT_OFFSET		0773
			25	11	001CF	BRB	288			
A1	8F		50	91	001D1	248:	CMPB	R0, #161		0775
			1F	12	001D5	BNEQ	288			
	50	08	A3	32	001D7	CVTWL	8(R3), R0			0777
18	BC	0A	50	C0	001DB	ADDL2	R0, @BIT_OFFSET			
	05		A3	E9	001DF	BLBC	10(R3), 258			0778
	65		2A	90	001E3	MOVB	#42, (R5)			0779
			0E	11	001E6	BRB	288			
	65		22	90	001E8	258:	MOVB	#34, (R5)		0780
			09	11	001EB	268:	BRB	288		0770
			8F	DD	001ED	278:	PUSHL	#165888		0788
	69	00028800	01	FB	001F3	CALLS	#1, LIB\$SIGNAL			

	20		65	91	001F6	28%:	CMPB	(R5), #32	0791	
			0A	12	001F9		BNEQ	29%		
			64	94	001FB		CLRB	(R4)	0794	
	65		17	90	001FD		MOVB	#23, (R5)	0795	
	62		02	80	00200		MOVW	#2, (R2)	0796	
			17	11	00203		BRB	30%	0797	
	21		65	91	00205	29%:	CMPB	(R5), #33	0799	
			1A	12	00208		BNEQ	31%		
			64	94	0020A		CLRB	(R4)	0802	
	65		16	90	0020C		MOVB	#22, (R5)	0803	
			7E	7C	0020F		CLRQ	-(SP)	0805	
		04	A2	DD	00211		PUSHL	4(R2)		
	68		03	FB	00214		CALLS	#3, DBG\$INS_DECODE		
62	50		04	A2	A3	00217	SUBW3	4(R2), R0, TR2	0806	
00000000G	00		08	A3	DD	0021C	30%:	MOVL	8(R3), DBG\$GL_CALL_CONTEXT	0807
			52	DD	00224	31%:	PUSHL	R2	0810	
	FAE6	CF	01	FB	00226		CALLS	#1, DBG\$DATA_LENGTH		
	14	BC	50	DD	00228		MOVL	R0, @BIT_LENGTH		
			22	11	0022F		BRB	33%	0615	
	52		10	AC	DD	00231	32%:	MOVL	VMS_DESC, R2	0821
	02	A2	16	80	00235		MOVW	#22, 2(R2)	0822	
	04	A2	04	B2	DD	00239	ADDL2	@4(R2), 4(R2)	0824	
			7E	7C	0023E		CLRQ	-(SP)	0826	
			04	A2	DD	00240	PUSHL	4(R2)		
	68		03	FB	00243		CALLS	#3, DBG\$INS_DECODE		
62	50		04	A2	A3	00246	SUBW3	4(R2), R0, TR2	0827	
	50		62	3C	00248		MOVZWL	(R2), R0	0828	
14	BC		03	78	0024E		ASHL	#3, R0, @BIT_LENGTH		
	50		01	DD	00253	33%:	MOVL	#1, R0	0837	
			04	DD	00256		RET		0838	

; Routine Size: 599 bytes, Routine Base: DBG\$CODE + 02EF

```

714 0839 1 GLOBAL ROUTINE DBGSMKE_VMS_DESC (prm_desc,vms_desc) =
715 0840 1
716 0841 1 FUNCTIONAL DESCRIPTION:
717 0842 1
718 0843 1 This routine constructs a VAX/VMS descriptor that points to the
719 0844 1 value of a symbol (described by a primary symbol descriptor).
720 0845 1 It first materializes the address by resolving all array and/or
721 0846 1 record component references, and then determines the length and
722 0847 1 data type. Finally it checks for sub references, and if one is
723 0848 1 present updates the length and address fields accordingly.
724 0849 1
725 0850 1 FORMAL PARAMETERS:
726 0851 1
727 0852 1 prm_desc - A longword containing the address of a primary descriptor
728 0853 1
729 0854 1 vms_desc - A longword containing the address of a block of at least
730 0855 1 12 bytes where a VAX/VMS descriptor will be constructed.
731 0856 1

```

```
733 0857 1 !dbg$make_vms_desc (prm_desc,vms_desc)
734 0858 BEGIN
735 0859 MAP
736 0860     prm_desc      : REF dbg$primary,
737 0861     vms_desc     : REF dbg$stg_desc;
738 0862
739 0863 LOCAL
740 0864     adr_kind,
741 0865     adr_ptr      : VECTOR [3, LONG],
742 0866     addr_offset,
743 0867     bit_offset,
744 0868     bit_length,
745 0869     result_desc  : BLOCK [12, BYTE],
746 0870     data_subnode : REF dbg$prim_node,
747 0871     prim_subnode : REF dbg$prim_node,
748 0872     typeid: REF rst$entry,
749 0873     s_value;
750 0874
751 0875 BUILTIN
752 0876 PROBER;
753 0877
754 0878 dbg$gl_current_primary = .prm_desc;          ! A003
755 0879
756 0880 ! It is illegal to call DBG$MAKE_VMS_DESC with a type.
757 0881
758 0882 IF .prm_desc[dbg$b_dhdr_kind] EQL rst$k_type
759 0883 THEN
760 0884 BEGIN
761 0885 LOCAL
762 0886     name;
763 0887 IF .prm_desc[dbg$l_dhdr_symid0] NEQ 0
764 0888 THEN
765 0889 BEGIN
766 0890     dbg$sta_symname(.prm_desc[dbg$l_dhdr_symid0], name);
767 0891     SIGNAL (dbg$_novaltyp, 1, .name);
768 0892 END
769 0893 ELSE
770 0894     SIGNAL (dbg$_novalue);
771 0895 END;
772 0896
773 0897 !+
774 0898 ! The first thing we do is to set the symbol table access context to
775 0899 ! the correct stack frame (in case of values whose address is given by
776 0900 ! an offset from an address in a register such as FP or AP), and clear
777 0901 ! the initial byte and bit addresses and descriptor fields.
778 0902
779 0903 dbg$sta_setcontext(.prm_desc[dbg$l_dhdr_symid0]);
780 0904 bit_offset = bit_length = 0;
781 0905 ch$fill(0,12,result_desc);
```



```
783 0906 2
784 0907 2
785 0908 2
786 0909 2
787 0910 2
788 0911 2
789 0912 2
790 0913 2
791 0914 2
792 0915 2
793 0916 2
794 0917 2
795 0918 2
796 0919 2
797 0920 2
798 0921 2
799 0922 2
800 0923 2
801 0924 2
802 0925 2
803 0926 2
804 0927 2
805 0928 2
806 0929 2
807 0930 2
808 0931 2
809 0932 2
810 0933 2
811 0934 2
812 0935 2
813 0936 2
814 0937 2
815 0938 2
816 0939 2
817 0940 2
818 0941 2
819 0942 2
820 0943 2
821 0944 2
822 0945 2
823 0946 2
824 0947 2
825 0948 2
826 0949 2
827 0950 2
828 0951 2

Loop for all except the last sub-node in the primary descriptor,
building up the address of the element within the structure.
This address is represented by a byte address, stored in the
dsc$a_pointer field of result_desc, and a bit offset, stored
(for now) in the local variable bit_offset.

N.B. This MUST be done 'top-down' to deal with POINTER data-types.

prim_subnode = .prm_desc[dbg$l_prim_flink];
data_subnode = .prm_desc[dbg$l_prim_blink];
WHILE .prim_subnode NEQA .data_subnode DO
  BEGIN
    result_desc[dsc$a_pointer] = .result_desc[dsc$a_pointer] + .prim_subnode[dbg$l_pnode_reloc];
    +
    Get the address of this element by calling dbg$sta_symvalue.
    This should return either the address of a data item (offset
    from the start of the record, if this is a record component)
    or the address of a descriptor. Anything else is an error.
    -
    IF (.prim_subnode[dbg$l_pnode_symid] NEQ 0) AND
    NOT .prim_subnode[dbg$u_pnode_ignore]
    THEN
      BEGIN
        dbg$sta_symvalue(.prim_subnode[dbg$l_pnode_symid],adr_ptrs,adr_kind);
        SELECTORE .adr_kind OF
          SET
            [dbg$k_val_addr]:
              BEGIN
                result_desc[dsc$a_pointer] = .result_desc[dsc$a_pointer] + .adr_ptrs[0];
                bit_offset = .bit_offset + .adr_ptrs[1];
              END;
            [dbg$k_val_descr]:
              BEGIN
                BIND adr_desc = adr_ptrs[0] : REF dbg$stg_desc;
                result_desc[dsc$a_pointer] = .result_desc[dsc$a_pointer] + .adr_desc[dsc$a_pointer];
              END;
            [dbg$k_val_unalloc]:
              BEGIN
                LOCAL name;
                dbg$sta_symname(.prim_subnode[dbg$l_pnode_symid],name);
                SIGNAL(dbg$unallocated, 1, .name);
              END;
          [OTHERWISE]:
            SIGNAL(dbg$_novalue);
        TES;
      END;
    END;
  END;
```

```
SELECTONE .prim_subnode[dbg$b_pnode_fcode] OF
SET
[rst$k_type_array]:
BEGIN
    +
    | If this is an array (of records or pointers, presumably),
    | get the address of an individual element of the array.
    -
    BIND psub = prim_subnode[dbg$a_pnarr_svector] : dbg$prim_node_subs;
    addr_offset = .prim_subnode[dbg$l_pnarr_offset];

    | Loop through the dimensions of the array.
    DECR index FROM .prim_subnode[dbg$b_pnarr_dimcnt]-1 TO 0 DO
    BEGIN
        s_value = .psub[.index,dbg$l_pnsub_svalue];
        typeid = .psub[.index,dbg$l_pnsub_typeid];

        | If the array is indexed by an enumeration type,
        | then index by the position and not by
        | the value. This case arises only in ADA for
        | enumerated types with representation specs.
        IF (.dbg$gb_language EQL dbg$k_ada) AND
            (.typeid NEQ 0)
        THEN
            IF (.typeid[rst$b_fcode] EQL rst$k_type_enum)
            THEN
                s_value = dbg$enum_pos(.typeid,.s_value);

            addr_offset = .addr_offset + (.s_value*.psub[.index,dbg$l_pnsub_stride]);
        END;

        IF .prim_subnode[dbg$b_pnarr_bitref]
        THEN bit_offset = .bit_offset + .addr_offset
        ELSE result_desc[dsc$a_pointer] = .result_desc[dsc$a_pointer] + .addr_offset;
        END;

[rst$k_type_ptr,rst$k_type_tptr]:
BEGIN
    BUILTIN PROBER;
    LOCAL addr;
    +
    | Check for read access before trying to fetch a value.
    -
    addr = .result_desc[dsc$a_pointer] + .bit_offset<3,29,1>;
    IF NOT PROBER(%REF(0),%REF(5),addr)
    THEN SIGNAL(dbg$noaccessr,1,.addr);
    +
    | If this is a POINTER, fetch a longword value
    -
    result_desc[dsc$a_pointer] = .(addr)<.bit_offset,32,0>;
    bit_offset = 0;
    END;

[rst$k_type_record]:
0;
```

```
[rst$sk_type_variant]:
  IF NOT .prim_subnode[dbg$vnvar_valid] THEN
    BEGIN
      LOCAL tag_value, tag_size, tag_name : REF VECTOR[.BYTE];
      prim_subnode[dbg$vnvar_valid] = true;
      IF .prim_subnode[dbg$lnvar_tagid] NEQ 0 THEN
        BEGIN
          BUILTIN PROBER;
          dbg$sta_symname(.prim_subnode[dbg$lnvar_tagid], tag_name);
          dbg$sta_symsize(.prim_subnode[dbg$lnvar_tagid], tag_size);
          IF (.tag_size NEQ 0) AND (.tag_name[0] NEQ 0) THEN
            BEGIN
              dbg$sta_symvalue(.prim_subnode[dbg$lnvar_tagid], adr_ptrs, adr_kind);
              adr_ptrs[0] = .adr_ptrs[0] + .result_desc[dsc$a_pointer];
              adr_ptrs[1] = .adr_ptrs[1] + .bit_offset;

              ++
              | Check that the address is accessible
              --
              IF NOT PROBER( %REF(0), %REF(4), .adr_ptrs[0] )
                THEN
                  SIGNAL(dbg$_noaccessr, 1, .adr_ptrs[0]);

              tag_value = (.adr_ptrs[0] < .adr_ptrs[1], .tag_size, 0);
              IF NOT dbg$sta_variant_value(.tag_value, .prim_subnode[dbg$lnvar_dstptr])
                THEN SIGNAL(dbg$_badtagval, 2, .tag_value, tag_name[0]);
            END;
          END;
        END;
      END;

[rst$sk_type_file]:
  BEGIN
    BUILTIN PROBER;
    LOCAL addr: REF BITVECTOR[];
    +
    | For file types what we have in the vms descriptor is a
    | pointer to a PASCAL file descriptor. Bit 16 in the
    | second longword of this descriptor is a "valid" bit which
    | basically says whether the file is open. If bit 16 is set
    | then the first longword of the descriptor is a pointer
    | to a buffer from which we can read the next item
    | in the file.
    -
    +
    | Note in the calculations below, bit_offset will normally
    | be zero. It might conceivably be non-zero in obscure cases,
    | such as a file variable which is an element of a packed
    | record.
    -
    +
    | Check for read access.
    -
    addr = .result_desc[dsc$a_pointer] + .bit_offset<3,29,1>;
    bit_offset = .bit_offset<0,3,0>;
    IF NOT PROBER(%REF(0), %REF(8), .addr) THEN SIGNAL(dbg$_illfilptr);
    +
```



```

: 944
: 945
: 946
: 947
: 948
: 949
: 950
: 951
: 952
: 953
: 954
: 955
: 956
: 957
: 958
: 959
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081

```

```

: Check "valid" bit.
: _
: IF NOT .addr[48+.bit_offset] THEN SIGNAL(dbg$_illfilptr);
: _
: Put the buffer address back into result_desc.
: _
: result_desc[dsc$a_pointer] = .(.addr)<.bit_offset,32,0>;
: bit_offset = 0;
: END;

[OTHERWISE]:
: SIGNAL(dbg$_illtype);

: TES;
prim_subnode = .prim_subnode[dbg$_l_pnode_flink];
: END;

```

```

961 1082 2
962 1083 2
963 1084 2
964 1085 2
965 1086 2
966 1087 2
967 1088 2
968 1089 2
969 1090 2
970 1091 2
971 1092 2
972 1093 2
973 1094 2
974 1095 2
975 1096 2
976 1097 2
977 1098 2
978 1099 2
979 1100 2
980 1101 2
981 1102 2
982 1103 2
983 1104 2
984 1105 2
985 1106 2
986 1107 2
987 1108 2
988 1109 2
989 1110 2
990 1111 2
991 1112 2
992 1113 2
993 1114 2
994 1115 2
995 1116 2
996 1117 2
997 1118 2

+
We now have resolved all earlier record component and array references,
and have to determine the actual data item address. The first thing to
do is to calculate the base address (much as we did above).
-
IF (.data_subnode[dbg$l_pnode_symid] NEQ 0) AND
NOT .data_subnode[dbg$v_pnode_ignore]
THEN
  BEGIN
    dbg$sta symvalue(.data_subnode[dbg$l_pnode_symid],adr_ptrs,adr_kind);
    SELECTORE .adr_kind OF
      SET
        [dbg$k_val_addr]:
          BEGIN
            result_desc[dsc$a_pointer] = .result_desc[dsc$a_pointer] + .adr_ptrs[0];
            bit_offset = .bit_offset + .adr_ptrs[1];
          END;
        [dbg$k_val_descr]:
          BEGIN
            BIND adr_desc = adr_ptrs[0] : REF dbg$stg_desc;
            result_desc[dsc$a_pointer] = .result_desc[dsc$a_pointer] + .adr_desc[dsc$a_pointer];
          END;
        [dbg$k_val_literal]:
          BEGIN
            result_desc[dsc$a_pointer] = .adr_ptrs[0];
            bit_offset = .adr_ptrs[1];
          END;
        [dbg$k_val_unalloc]:
          BEGIN
            LOCAL name;
            dbg$sta symname(.data_subnode[dbg$l_pnode_symid],name);
            SIGNAL(dbg$_unallocated, 1, .name);
          END;
      [OTHERWISE]:
        SIGNAL(dbg$_novalue);
    TES;
  END;
```

```

999      1119  2
1000     1120  2
1001     1121  2
1002     1122  2
1003     1123  2
1004     1124  2
1005     1125  2
1006     1126  2
1007     1127  2
1008     1128  2
1009     1129  2
1010     1130  2
1011     1131  3
1012     1132  3
1013     1133  3
1014     1134  3
1015     1135  4
1016     1136  4
1017     1137  4
1018     1138  4
1019     1139  4
1020     1140  4
1021     1141  3
1022     1142  4
1023     1143  4
1024     1144  4
1025     1145  4
1026     1146  4
1027     1147  4
1028     1148  4
1029     1149  3
1030     1150  4
1031     1151  4
1032     1152  4
1033     1153  4
1034     1154  4
1035     1155  4
1036     1156  4
1037     1157  4
1038     1158  4
1039     1159  5
1040     1160  5
1041     1161  5
1042     1162  5
1043     1163  4
1044     1164  5
1045     1165  5
1046     1166  6
1047     1167  5
1048     1168  4
1049     1169  3
1050     1170  2
1051     1171  2
1052     1172  2
1053     1173  2
1054     1174  3
1055     1175  3

+
Having determined the address of the data object, we now attempt to
fill in the rest of the fields in the VMS descriptor (class, dtype,
and length). We use the information that we have in the bottom subnode:
a kind, an fcode, and a typeid.
CASE .data_subnode[dbg$b_pnode_kind] FROM rst$k_kind_minimum TO rst$k_kind_maximum OF
SET
  [rst$k_routine,rst$k_block,rst$k_entry,rst$k_line,rst$k_label]:
  BEGIN
  +
  Special case for MACRO - since MACRO declares everything to
  be a label, then we want to instead use the default type
  that has been specified with a SET TYPE command.
  IF (.data_subnode[dbg$b_pnode_kind] EQL rst$k_label)
  AND (.prm_desc[dbg$b_dhdr_lang] EQL dbg$k_macro)
  AND (NOT .prm_desc[dbg$b_dhdr_override])
  AND (.dbg$gl_dflttyp NEQ dsc$k_dtype_zi)      ! If instruction, then we
                                                ! already have correct type
                                                ! and length
  THEN
  BEGIN
    result_desc[dsc$b_class] = dsc$k_class_z;
    result_desc[dsc$b_dtype] = .dbg$gl_dflttyp;
    result_desc[dsc$b_length] = .dbg$gw_dfltleng; ! All default types
    bit_length = 8*.dbg$gw_dfltleng;             ! have length in bytes.
  END
  ELSE
  BEGIN
  +
  This must be a primary representing an instruction or an entry
  mask in the user program.
  IF .bit_offset NEQ 0 THEN SIGNAL(dbg$unimplent);
  result_desc[dsc$b_class] = dsc$k_class_z;
  IF dbg$is_it_entry(.result_desc[dsc$a_pointer])
  THEN
  BEGIN
    result_desc[dsc$b_dtype] = dsc$k_dtype_zem;
    bit_length = 16;
  END
  ELSE
  BEGIN
    result_desc[dsc$b_dtype] = dsc$k_dtype_zi;
    bit_length = %BPUNIT*(dbg$ins_decode(.result_desc[dsc$a_pointer],false,false) -
                                .result_desc[dsc$a_pointer]);
  END;
  END;
END;

[rst$k_data,rst$k_ttypcomp]:
BEGIN
+
The Primary represents data in the user program. Note that
```

1056 1176 3
1057 1177 3
1058 1178 3
1059 1179 3
1060 1180 4
1061 1181 4
1062 1182 4
1063 1183 4
1064 1184 4
1065 1185 4
1066 1186 4
1067 1187 4
1068 1188 5
1069 1189 5
1070 1190 5
1071 1191 5
1072 1192 5
1073 1193 5
1074 1194 5
1075 1195 5
1076 1196 5
1077 1197 5
1078 1198 6
1079 1199 5
1080 1200 6
1081 1201 5
1082 1202 5
1083 1203 5
1084 1204 5
1085 1205 4
1086 1206 4
1087 1207 4
1088 1208 4
1089 1209 4
1090 1210 4
1091 1211 4
1092 1212 4
1093 1213 4
1094 1214 4
1095 1215 4
1096 1216 4
1097 1217 4
1098 1218 4
1099 1219 4
1100 1220 4
1101 1221 4
1102 1222 5
1103 1223 5
1104 1224 6
1105 1225 5
1106 1226 6
1107 1227 6
1108 1228 6
1109 1229 6
1110 1230 6
1111 1231 5
1112 1232 5

```
! record components come back with kind=typcomp (this is a quirk
! in the parsing that may eventually be fixed).
IF .data_subnode[dbg$b_pnode_fcode] EQL rst$k_type_array
THEN
  BEGIN
    BIND pnsb = data_subnode[dbg$a_pnarr_svector] : dbg$prim_node_subs;
    addr_offset = .data_subnode[dbg$l_pnarr_offset];

    ! Loop through the dimensions of the array.
    DECR index FROM .data_subnode[dbg$b_pnarr_dimcnt]-1 TO 0 DO
      BEGIN
        s_value = .pnsb[.index,dbg$l_pnsb_svalue];
        typeid = .pnsb[.index,dbg$l_pnsb_typeid];

        ! If the array is indexed by an enumeration type,
        ! then index by the position and not by
        ! the value. This cases arises only in ADA for
        ! enumerated types with representation specs.
        IF (.dbg$gb_language EQL dbg$k_ada) AND
            (.typeid NEQ 0)
        THEN
          IF (.typeid[rst$b_fcode] EQL rst$k_type_enum)
          THEN
            s_value = dbg$enum_pos(.typeid,.s_value);

            addr_offset = .addr_offset + (.s_value*.pnsb[.index,dbg$l_pnsb_stride]);
          END;
        ELSE
          IF .data_subnode[dbg$b_pnarr_bitref]
          THEN bit_offset = .bit_offset + .addr_offset
          ELSE result_desc[dsc$a_pointer] = .result_desc[dsc$a_pointer] + .addr_offset;

          !+
          ! Figure out the class field. This class may get fixed up later -
          ! we want class=ubs to reflect the fact that there is a bit
          ! offset present. But for now, we just fill in class VS for
          ! dtype vt, class SD if digits or scale are present, and
          ! class S for all other dtypes.
          IF .data_subnode[dbg$b_pnarr_dtype] EQL dsc$k_dtype_vt
          THEN
            result_desc[dsc$b_class] = dsc$k_class_vs
          ELSE
            BEGIN
              IF (.data_subnode[dbg$b_pnarr_digits] NEQ 0) OR
                  (.data_subnode[dbg$b_pnarr_scale] NEQ 0)
              THEN
                BEGIN
                  result_desc[dsc$b_class] = dsc$k_class_sd;
                  result_desc[dsc$b_digits] = .data_subnode[dbg$b_pnarr_digits];
                  result_desc[dsc$b_scale] = .data_subnode[dbg$b_pnarr_scale];
                END
              ELSE
                result_desc[dsc$b_class] = dsc$k_class_s;
```



```
1113 1233 4
1114 1234 4
1115 1235 4
1116 1236 4
1117 1237 4
1118 1238 4
1119 1239 4
1120 1240 4
1121 1241 4
1122 1242 4
1123 1243 4
1124 1244 4
1125 1245 4
1126 1246 5
1127 1247 5
1128 1248 5
1129 1249 5
1130 1250 5
1131 1251 5
1132 1252 5
1133 1253 5
1134 1254 4
1135 1255 4
1136 1256 4
1137 1257 5
1138 1258 5
1139 1259 5
1140 1260 5
1141 1261 5
1142 1262 5
1143 1263 5
1144 1264 5
1145 1265 5
1146 1266 5
```

```
END;

! Fill in dtype and length. For arrays, we have this information
! in the array subnode.
result_desc[dsc$b_dtype] = .data_subnode[dbg$b_pnarr_dtype];
result_desc[dsc$w_length] = .data_subnode[dbg$w_pnarr_length];
!
! Fix up boolean to use the new type code dsc$k_dtype_tf.
IF .result_desc[dsc$b_dtype] EQL dsc$k_bool
THEN
  BEGIN
    result_desc[dsc$b_class] = dsc$k_class_s; ! This will get
                                                ! changed to UBS
                                                ! if there is a
                                                ! bit offset
    result_desc[dsc$b_dtype] = dsc$k_dtype_tf;
    result_desc[dsc$w_length] = bit_length = 1;
  END
ELSE
  bit_length = dbg$data_length(result_desc);
END
ELSE
  !
  ! For everything except arrays, we call a routine to fill
  ! in the fields of the VMS descriptor, passing to this
  ! routine all the information we have collected so far.
  dbg$fill_in_vms_desc(.data_subnode[dbg$b_pnode_fcode],
    .data_subnode[dbg$l_pnode_typeid],
    .prm_desc[dbg$l_dhdr_symid0],
    result_desc, bit_length, bit_offset);
```

```
1148 1267
1149 1268
1150 1269
1151 1270
1152 1271
1153 1272
1154 1273
1155 1274
1156 1275
1157 1276
1158 1277
1159 1278
1160 1279
1161 1280
1162 1281
1163 1282
1164 1283
1165 1284
1166 1285
1167 1286
1168 1287
1169 1288
1170 1289
1171 1290
1172 1291
1173 1292
1174 1293
1175 1294
1176 1295
1177 1296
1178 1297
1179 1298
1180 1299
1181 1300
1182 1301
1183 1302
1184 1303
1185 1304
1186 1305
1187 1306
1188 1307
1189 1308
1190 1309
1191 1310
1192 1311
1193 1312
1194 1313
1195 1314
1196 1315
1197 1316
1198 1317
1199 1318

+
+ Fix things up so class VS always has dtype VT. Class VS and
+ type T is the older way of expressing varying string data
+ type, so we fix it up to use the newer dtype VT.
-
IF .result_desc[dsc$b_class] EQL dsc$k_class_vs
AND .result_desc[dsc$b_dtype] EQL dsc$k_dtype_t
THEN result_desc[dsc$b_dtype] = dsc$k_dtype_vt;
END;

+
+ We should not see other kinds.
-
[INRANGE,OUTRANGE]:
SIGNAL(dbg$_unimplent);
TES;

! Dereference descriptors of type DSC.
IF .result_desc[dsc$b_dtype] EQL dsc$k_dtype_dsc
THEN
BEGIN
IF NOT PROBER(%REF(0),%REF(8),.result_desc[dsc$a_pointer])
THEN
SIGNAL(dbg$_noaccessr,1,.result_desc[dsc$a_pointer]);
ch$move(.result_desc[dsc$b_length],
.result_desc[dsc$a_pointer], result_desc);
END;

! Dereference descriptors of type BPV or BLV.
IF (.result_desc[dsc$b_dtype] EQL dsc$k_dtype_blv) OR
(.result_desc[dsc$b_dtype] EQL dsc$k_dtype_bpv)
THEN
BEGIN
IF NOT PROBER(%REF(0),%REF(8),.result_desc[dsc$a_pointer])
THEN
SIGNAL(dbg$_noaccessr,1,.result_desc[dsc$a_pointer]);
result_desc[dsc$a_pointer] = .result_desc[dsc$a_pointer];
IF .result_desc[dsc$b_dtype] EQL dsc$k_dtype_blv
THEN
result_desc[dsc$b_dtype] = dsc$k_dtype_zi;
IF .result_desc[dsc$b_dtype] EQL dsc$k_dtype_bpv
THEN
result_desc[dsc$b_dtype] = dsc$k_dtype_zem;
END;
result_desc[dsc$a_pointer] =
.result_desc[dsc$a_pointer] + .data_subnode[dbg$_pnode_reloc];
```

```
1201 1319 2 IF .prm_desc[dbg$u_dhdr_subref] THEN
1202 1320 BEGIN
1203 1321
1204 1322     If there was an offset in the Primary (either a bit offset or
1205 1323     a byte offset) then take care of that here.
1206 1324
1207 1325 IF .prm_desc[dbg$u_dhdr_bitref]
1208 1326 THEN
1209 1327 BEGIN
1210 1328     bit_offset = .prm_desc[dbg$u_prim_offset] + .bit_offset;
1211 1329     bit_length = .prm_desc[dbg$u_prim_length];
1212 1330     result_desc[dsc$b_class] = dsc$k_class_z;    ! These get fixed up
1213 1331     result_desc[dsc$b_dtype] = dsc$k_dtype_z;    ! below.
1214 1332     result_desc[dsc$w_length] = 0;
1215 1333     IF (.bit_offset AND (%BPUNIT-1)) EQL 0 THEN
1216 1334         SELECT ONE .bit_length OF
1217 1335         SET
1218 1336             [ 8]:
1219 1337                 result_desc[dsc$b_dtype] = (IF .prm_desc[dbg$u_dhdr_sgnext]
1220 1338                 THEN dsc$k_dtype_b ELSE dsc$k_dtype_bu);
1221 1339             [16]:
1222 1340                 result_desc[dsc$b_dtype] = (IF .prm_desc[dbg$u_dhdr_sgnext]
1223 1341                 THEN dsc$k_dtype_w ELSE dsc$k_dtype_wu);
1224 1342             [32]:
1225 1343                 result_desc[dsc$b_dtype] = (IF .prm_desc[dbg$u_dhdr_sgnext]
1226 1344                 THEN dsc$k_dtype_l ELSE dsc$k_dtype_lu);
1227 1345
1228 1346             [OTHERWISE]:
1229 1347                 0;
1230 1348
1231 1349         TES;
1232 1350     END
1233 1351 ELSE
1234 1352 BEGIN
1235 1353     IF .result_desc[dsc$b_dtype] EQL dsc$k_dtype_vt
1236 1354     THEN
1237 1355         BEGIN
1238 1356             result_desc[dsc$b_class] = dsc$k_class_s;
1239 1357             result_desc[dsc$b_dtype] = dsc$k_dtype_t;
1240 1358             result_desc[dsc$a_pointer] = .result_desc[dsc$a_pointer] + 2;
1241 1359         END;
1242 1360     result_desc[dsc$a_pointer] = .prm_desc[dbg$u_prim_offset] + .result_desc[dsc$a_pointer];
1243 1361     bit_length = .prm_desc[dbg$u_prim_length] * %BPUNIT;
1244 1362
1245 1363     !
1246 1364     ! For the string data types (which include ascii and also the
1247 1365     ! numeric string types NU, NL, NLO, NR, NRO, and NZ), fill
1248 1366     ! in the length of the result descriptor, but leave the class
1249 1367     ! and dtype unchanged. Note that the code below relies on the
1250 1368     ! fact that these dtype codes span the range from 14 (dsc$k_dtype_t)
1251 1369     ! to 20 (dsc$k_dtype_nz).
1252 1370
1253 1371     IF (.result_desc[dsc$b_dtype] GEQ dsc$k_dtype_t) AND
1254 1372     (.result_desc[dsc$b_dtype] LEQ dsc$k_dtype_nz)
1255 1373     THEN
1256 1374
1257 1375
```

DBGVALUES
V04-000

G 13
16-Sep-1984 02:45:26
14-Sep-1984 12:17:54

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGVALUES.B32;1

Page 38
(16)

1258	1376	4
1259	1377	4
1260	1378	3
1261	1379	3
1262	1380	3
1263	1381	3
1264	1382	4
1265	1383	3
1266	1384	2

```
result_desc[dsc$w_length] = .prm_desc[dbg$w_prim_length]
ELSE
BEGIN
result_desc[dsc$b_class] = dsc$k_class_z;      ! (0)
result_desc[dsc$b_dtype] = dsc$k_dtype_z;      ! (0)
result_desc[dsc$w_length] = 0;
END;
END;
END;
```


1268	1385	
1269	1386	
1270	1387	
1271	1388	
1272	1389	
1273	1390	
1274	1391	
1275	1392	
1276	1393	
1277	1394	
1278	1395	
1279	1396	
1280	1397	
1281	1398	
1282	1399	
1283	1400	
1284	1401	
1285	1402	
1286	1403	
1287	1404	
1288	1405	
1289	1406	
1290	1407	
1291	1408	
1292	1409	
1293	1410	
1294	1411	
1295	1412	
1296	1413	
1297	1414	
1298	1415	
1299	1416	
1300	1417	
1301	1418	
1302	1419	
1303	1420	
1304	1421	
1305	1422	
1306	1423	
1307	1424	
1308	1425	
1309	1426	
1310	1427	
1311	1428	
1312	1429	
1313	1430	
1314	1431	
1315	1432	
1316	1433	
1317	1434	
1318	1435	
1319	1436	
1320	1437	
1321	1438	
1322	1439	
1323	1440	
1324	1441	

```
At this point, if the bit_offset variable is not a multiple of 8 then
there really is a bit offset. Fix up the class field to be UBS in
this case.
IF (.bit_offset AND (%BPUNIT-1)) NEQ 0 THEN
  BEGIN
    We used to not support unaligned bit fields longer than 32 bits.
    IF .bit_length GTRU 32 THEN SIGNAL(dbg$_unimplent);
    result_desc[dsc$b_class] = dsc$k_class_ubs;
    IF .result_desc[dsc$b_dtype] EQL dsc$k_dtype_z
    THEN result_desc[dsc$b_dtype] = dsc$k_dtype_vu;
    Bit length is in bits for these five data types, and in bytes
    for all others.
    result_desc[dsc$w_length] = .bit_length /
      (IF .result_desc[dsc$b_dtype] EQL dsc$k_dtype_vu
      OR .result_desc[dsc$b_dtype] EQL dsc$k_dtype_v
      OR .result_desc[dsc$b_dtype] EQL dsc$k_dtype_svu
      OR .result_desc[dsc$b_dtype] EQL dsc$k_dtype_sv
      OR .result_desc[dsc$b_dtype] EQL dsc$k_dtype_tf
      THEN 1 ELSE 8);
    result_desc[dsc$l_pos] = .bit_offset;
  END
ELSE
  BEGIN
    If we get here then we have byte-aligned data.
    Fix up the pointer field to point to the byte where the data
    data actually begins.
    result_desc[dsc$a_pointer] = .result_desc[dsc$a_pointer] + (.bit_offset/%BPUNIT);
    IF (.result_desc[dsc$b_class] EQL dsc$k_class_z)
    THEN
      BEGIN
        IF ((.bit_length AND (%BPUNIT-1)) EQL 0)
        THEN
          If the length of the data is exactly a multiple of 8 then
          leave the dtype Z and express the length in bytes.
          result_desc[dsc$w_length] = .bit_length/%BPUNIT
        ELSE
          BEGIN
            If the length is not expressible in bytes then change the dtype
            to V and fill in the length field with a bit length.
            result_desc[dsc$b_dtype] = dsc$k_dtype_v;
            IF .bit_length LSSU '10000'
            THEN
              BEGIN
                result_desc[dsc$w_length] = .bit_length;
                result_desc[dsc$l_pos] = 0;
```

1325 1442 00
1326 1443 00
1327 1444 00
1328 1445 00
1329 1446 00
1330 1447 00
1331 1448 00
1332 1449 00
1333 1450 00
1334 1451 00
1335 1452 00
1336 1453 00
1337 1454 00
1338 1455 00
1339 1456 00
1340 1457 00
1341 1458 00
1342 1459 00
1343 1460 00

```
END
ELSE
  BEGIN
    Special handling for the case where the length does
    not fit in the word field.
    result_desc[dsc$w_length] = 0;
    result_desc[dsc$l_pos] = .bit_length;
  END;
END;
END;
END;
ch$move(12,result_desc,.vms_desc);
RETURN sts$ok_success;
END;
! End of routine dbg$make_vms_desc
```

			OFFC 00000		.ENTRY	DBG\$MAKE VMS_DESC. Save R2,R3,R4,R5,R6,R7,-	
	SB	00000000G	00	9E	00002	R8,R9,R10,R11	0839
	5E		38	C2	00009	LIB\$SIGNAL, R11	
	57	04	AC	D0	0000C	#56, SP	
00000000G	00		57	D0	00010	PRM_DESC, R7	0878
	5A	04	A7	9E	00017	R7, DBG\$GL_CURRENT_PRIMARY	
	07	03	AA	91	0001B	4(R7), R10	0882
			29	12	0001F	3(R10), #7	
		0C	A7	D5	00021	2\$	
			1B	13	00024	12(R7)	0887
			5E	DD	00026	1\$	
		0C	A7	DD	00028	SP	0890
00000000G	00		02	FB	0002B	12(R7)	
			6E	DD	00032	CALLS #2, DBG\$STA_SYMNAME	
			01	DD	00034	PUSHL NAME	0891
		00028168	8F	DD	00036	#1	
	6B		03	FB	0003C	#164200	
			09	11	0003F	CALLS #3, LIB\$SIGNAL	
		000287F8	8F	DD	00041	2\$	0887
			01	FB	00047	#165880	0894
			A7	DD	0004A	CALLS #1, LIB\$SIGNAL	
00000000G	00	0C	01	FB	0004D	12(R7)	0903
			AE	7C	00054	CALLS #1, DBG\$STA_SETCONTEXT	
0C	00		00	2C	00057	CLRQ BIT_OFFSET	0904
			AE		0005C	MOVCS #0, (SP), #0, #12, RESULT_DESC	0905
		20	A7	D0	0005E		
	53	14	A7	D0	00062	MOVL 20(R7), PRIM_SUBNODE	0914
	56	18	A7	D0	00062	MOVL 24(R7), DATA_SUBNODE	0915
	56		53	D1	00066	CMPL PRIM_SUBNODE, DATA_SUBNODE	0916
			03	12	00069	4\$	
			01E5	31	0006B	BNEQ 25\$	
24	AE	14	A3	C0	0006E	BRW 25\$	
	52	10	A3	D0	00073	ADDL2 20(PRIM_SUBNODE), RESULT_DESC+4	0918
						MOVL 16(PRIM_SUBNODE), R2	0925

5E	0A	A3	10	63	13	00077	BEQL	8\$		
			30	05	E0	00079	BBS	#5, 10(PRIM_SUBNODE), 8\$	0926	
				AE	9F	0007E	PUSHAB	ADR_KIND	0929	
				AE	9F	00081	PUSHAB	ADR_PTRS		
00000000G	00			52	DD	00084	PUSHL	R2		
	50		10	03	FB	00086	CALLS	#3, DBG\$STA_SYMVALUE		
	02			AE	D0	0008D	MOVL	ADR_KIND, R0	0930	
				50	D1	00091	CMPL	R0, #2	0932	
				0C	12	00094	BNEQ	5\$		
24	AE		2C	AE	C0	00096	ADDL2	ADR_PTRS, RESULT_DESC+4	0934	
18	AE		30	AE	C0	0009B	ADDL2	ADR_PTRS+4, BIT_OFFSET	0935	
				3A	11	000A0	BRB	8\$	0930	
	03			50	D1	000A2	CMPL	R0, #3	0937	
				0B	12	000A5	BNEQ	6\$		
	50		2C	AE	D0	000A7	MOVL	ADR_DESC, R0	0940	
24	AE		04	A0	C0	000AB	ADDL2	4(R0), RESULT_DESC+4		
				2A	11	000B0	BRB	8\$	0930	
	04			50	D1	000B2	CMPL	R0, #4	0942	
				1C	12	000B5	BNEQ	7\$		
			04	AE	9F	000B7	PUSHAB	NAME	0945	
				52	DD	000BA	PUSHL	R2		
00000000G	00		04	02	FB	000BC	CALLS	#2, DBG\$STA_SYMNAME	0946	
				01	DD	000C6	PUSHL	#1		
		00028170		8F	DD	000C8	PUSHL	#164208		
	6B			03	FB	000CE	CALLS	#3, LIB\$SIGNAL		
		000287F8		09	11	000D1	BRB	8\$	0930	
				8F	DD	000D3	PUSHL	#165880	0949	
	6B			01	FB	000D9	CALLS	#1, LIB\$SIGNAL		
	50		09	A3	9A	000DC	MOVZBL	9(PRIM_SUBNODE), R0	0952	
	01			50	91	000E0	CMPL	R0, #1	0954	
				5D	12	000E3	BNEQ	14\$		
	58		20	A3	D0	000E5	MOVL	32(PRIM_SUBNODE), ADDR_OFFSET	0961	
	52		18	A3	9A	000E9	MOVZBL	27(PRIM_SUBNODE), INDEX	0965	
				3E	11	000ED	BRB	11\$		
54				14	C5	000EF	MULL3	#20, INDEX, R4	0967	
	52		28	A344	9F	000F3	PUSHAB	40(PRIM_SUBNODE)[R4]		
				9E	D0	000F7	MOVL	@(SP)+, S_VALUE		
	59		38	A344	9F	000FA	PUSHAB	56(PRIM_SUBNODE)[R4]	0968	
				9E	D0	000FE	MOVL	@(SP)+, TYPEID		
	55			00	91	00101	CMPL	DBG\$GB_LANGUAGE, #9	0975	
	09	00000000G		18	12	00108	BNEQ	10\$		
				55	D5	0010A	TSTL	TYPEID	0976	
				14	13	0010C	BEQL	10\$		
			04	A5	91	0010E	CMPL	24(TYPEID), #4	0978	
				0E	12	00112	BNEQ	10\$		
		0220		8F	BB	00114	PUSHR	#*M<R5, R9>	0980	
00000000G	00			02	FB	00118	CALLS	#2, DBG\$ENUM_POS		
	59			50	D0	0011F	MOVL	R0, S_VALUE		
			2C	A344	9F	00122	PUSHAB	44(PRIM_SUBNODE)[R4]	0982	
				9E	C5	00126	MULL3	@(SP)+, S_VALUE, R0		
50				50	C0	0012A	ADDL2	R0, ADDR_OFFSET		
	58			52	F4	0012D	SOBGEQ	INDEX, 9\$	0965	
	BF			02	E1	00130	BBC	#2, 10(PRIM_SUBNODE), 12\$	0985	
06	0A	A3		58	C0	00135	ADDL2	ADDR_OFFSET, BIT_OFFSET	0986	
	18	AE		04	11	00139	BRB	13\$		
				58	C0	0013B	ADDL2	ADDR_OFFSET, RESULT_DESC+4	0987	
	24	AE								

PC	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418	Op419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

		52	18	AE	FD	3E	12	00204	BNEQ	23\$		
				52	24	8F	78	00206	ASHL	#-3, BIT_OFFSET, ADDR	1062	
18	AE	18	AE	03		AE	CO	0020C	ADDL2	RESULT_DESC+4, ADDR		
		62		08		00	EF	00210	EXTZV	#0, #3, BIT_OFFSET, BIT_OFFSET	1063	
						00	OC	00217	PROBER	#0, #8, (ADDR)	1064	
					00028F40	09	12	0021B	BNEQ	21\$		
				6B		8F	DD	0021D	PUSHL	#167744		
		50	18	AE		01	FB	00223	CALLS	#1, LIB\$SIGNAL		
		09		62		30	C1	00226	ADDL3	#48, BIT_OFFSET, R0	1068	
					00028F40	50	E0	0022B	BBS	R0, (ADDR), 22\$		
				6B		8F	DD	0022F	PUSHL	#167744		
24	AE	62		20		01	FB	00235	CALLS	#1, LIB\$SIGNAL		
					18	AE	EF	00238	EXTZV	BIT_OFFSET, #32, (ADDR), RESULT_DESC+4	1072	
					18	AE	D4	0023F	CLRL	BIT_OFFSET	1073	
					000287D8	09	11	00242	BRB	24\$	0952	
				6B		8F	DD	00244	PUSHL	#165848	1077	
				53		01	FB	0024A	CALLS	#1, LIB\$SIGNAL		
						63	D0	0024D	MOVL	(PRIM_SUBNODE), PRIM_SUBNODE	1080	
				52	10	FE13	31	00250	BRW	3\$	0916	
						A6	D0	00253	MOVL	16(DATA_SUBNODE), R2	1087	
						74	13	00257	BEQL	30\$		
		6F	0A	A6		05	E0	00259	BBS	#5, 10(DATA_SUBNODE), 30\$	1088	
						AE	9F	0025E	PUSHAB	ADR_KIND	1091	
						AE	9F	00261	PUSHAB	ADR_PTRS		
						52	DD	00264	PUSHL	R2		
		00000000G		00		03	FB	00266	CALLS	#3, DBG\$STA_SYMVALUE		
				50	10	AE	D0	0026D	MOVL	ADR_KIND, R0	1092	
				02		50	D1	00271	CMPL	R0, #2	1094	
						OC	12	00274	BNEQ	26\$		
			24	AE	2C	AE	CO	00276	ADDL2	ADR_PTRS, RESULT_DESC+4	1096	
			18	AE	30	AE	CO	0027B	ADDL2	ADR_PTRS+4, BIT_OFFSET	1097	
						4B	11	00280	BRB	30\$	1092	
				03		50	D1	00282	CMPL	R0, #3	1099	
						0B	12	00285	BNEQ	27\$		
				50	2C	AE	D0	00287	MOVL	ADR_DESC, R0	1102	
			24	AE	04	A0	CO	0028B	ADDL2	4(R0), RESULT_DESC+4		
						3B	11	00290	BRB	30\$	1092	
				D1		50	D1	00292	CMPL	R0, #1	1104	
						OC	12	00295	BNEQ	28\$		
			24	AE	2C	AE	D0	00297	MOVL	ADR_PTRS, RESULT_DESC+4	1106	
			18	AE	30	AE	D0	0029C	MOVL	ADR_PTRS+4, BIT_OFFSET	1107	
						2A	11	002A1	BRB	30\$	1092	
				04		50	D1	002A3	CMPL	R0, #4	1109	
						1C	12	002A6	BNEQ	29\$		
					14	AE	9F	002A8	PUSHAB	NAME	1112	
						52	DD	002AB	PUSHL	R2		
		00000000G		00		02	FB	002AD	CALLS	#2, DBG\$STA_SYMNAME	1113	
						AE	DD	002B4	PUSHL	NAME		
						01	DD	002B7	PUSHL	#1		
					00028170	8F	DD	002B9	PUSHL	#164208		
				6B		03	FB	002BF	CALLS	#3, LIB\$SIGNAL	1092	
					000287F8	09	11	002C2	BRB	30\$	1116	
						8F	DD	002C4	PUSHL	#165880		
				6B		01	FB	002CA	CALLS	#1, LIB\$SIGNAL		
						A6	8F	002CD	CASEB	8(DATA_SUBNODE), #0, #13	1125	
0027		0D		00	08	001C		002D2	.WORD	32\$-31\$,-		
001C		0027		001C		0027		002DA		32\$-31\$,-		
		0098		0027								

1281
1143
1135
1136
1137
1138
1144
1145
1146
1155
1157
1160
1161
1157
1165
1166
1167
1166
1125
1179
1183
1187
1189
1190

		09	00000000G	00	91	00393	CMPB	DBG\$GB_LANGUAGE, #9	1197
				18	12	0039A	BNEQ	42\$	1198
				55	D5	0039C	TSTL	TYPEID	1200
				14	13	0039E	BEQL	42\$	1202
		04	18	A5	91	003A0	CMPB	24(TYPEID), #4	1204
				0E	12	003A4	BNEQ	42\$	1187
			0220	8F	BB	003A6	PUSHR	#M<R5,R9>	1207
				02	FB	003AA	CALLS	#2, DBG\$ENUM_POS	1208
				50	D0	003B1	MOVL	R0, S_VALUE	1209
				A6	9F	003B4	PUSHAB	44(DATA_SUBNODE)[R3]	1218
50				9E	C5	003B8	MULL3	@(SP)+, S_VALUE, R0	1220
				50	C0	003BC	ADDL2	R0, ADDR_OFFSET	1223
				52	F4	003BF	SOBGEQ	INDEX, 4T\$	1224
06	0A	A6		02	E1	003C2	BBC	#2, 10(DATA_SUBNODE), 44\$	1227
	18	AE		58	C0	003C7	ADDL2	ADDR_OFFSET, BIT_OFFSET	1229
				04	11	003CB	BRB	45\$	1232
	24	AE		58	C0	003CD	ADDL2	ADDR_OFFSET, RESULT_DESC+4	1233
	25		02	A4	91	003D1	CMPB	2(R4), #37	1239
				06	12	003D5	BNEQ	46\$	1240
	23	AE		0B	90	003D7	MOVB	#11, RESULT_DESC+3	1244
				17	11	003DB	BRB	49\$	1252
			01	A4	95	003DD	TSTB	1(R4)	1255
				04	12	003E0	BNEQ	47\$	1179
				64	95	003E2	TSTB	(R4)	1263
				0A	13	003E4	BEQL	48\$	1265
	23	AE		09	90	003E6	MOVB	#9, RESULT_DESC+3	1266
	28	AE		64	B0	003EA	MOVW	(R4), RESULT_DESC+8	1267
				04	11	003EE	BRB	49\$	1268
	23	AE		01	90	003F0	MOVB	#1, RESULT_DESC+3	1269
	22	AE	02	A4	90	003F4	MOVB	2(R4), RESULT_DESC+2	1270
	20	AE	1C	A6	B0	003F9	MOVW	28(DATA_SUBNODE), RESULT_DESC	1272
	9E	8F	22	AE	91	003FE	CMPB	RESULT_DESC+2, #15B	1273
				DE	12	00403	BNEQ	50\$	1274
	1C	AE		01	D0	00405	MOVL	#1, BIT_LENGTH	1277
	20	AE	01280001	8F	D0	00409	MOVL	#19398657, RESULT_DESC	1278
				26	11	00411	BRB	52\$	1279
			20	AE	9F	00413	PUSHAB	RESULT_DESC	1280
F69F	CF			01	FB	00416	CALLS	#1, DBG\$DATA_LENGTH	1179
1C	AE			50	D0	0041B	MOVL	R0, BIT_LENGTH	1263
				18	11	0041F	BRB	52\$	1264
			18	AE	9F	00421	PUSHAB	BIT_OFFSET	1265
			20	AE	9F	00424	PUSHAB	BIT_LENGTH	1266
			28	AE	9F	00427	PUSHAB	RESULT_DESC	1267
			0C	A7	DD	0042A	PUSHL	12(R7)	1268
			0C	A6	DD	0042D	PUSHL	12(DATA_SUBNODE)	1269
	7E		09	A6	9A	00430	MOVZBL	9(DATA_SUBNODE), -(SP)	1270
F970	CF			06	FB	00434	CALLS	#6, DBG\$FILL_IN_VMS_DESC	1272
	0B		23	AE	91	00439	CMPB	RESULT_DESC+3, #11	1273
				0A	12	0043D	BNEQ	53\$	1274
	0E		22	AE	91	0043F	CMPB	RESULT_DESC+2, #14	1277
				04	12	00443	BNEQ	53\$	1278
	22	AE		25	90	00445	MOVB	#37, RESULT_DESC+2	1279
	18		22	AE	91	00449	CMPB	RESULT_DESC+2, #24	1280
				1C	12	0044D	BNEQ	55\$	1281
24	BE	08		00	0C	0044F	PROBER	#0, #8, @RESULT_DESC+4	1290
				0E	12	00454	BNEQ	54\$	1292
			24	AE	DD	00456	PUSHL	RESULT_DESC+4	1292

			00028228	01	DD	00459	PUSHL	#1			
				8F	DD	0045B	PUSHL	#164392			
				03	FB	00461	CALLS	#3, LIB\$SIGNAL			
20	AE	24	BE	20	AE	28	00464	54\$:	MOVCL	RESULT_DESC, @RESULT_DESC+4, RESULT_DESC	1293
			21	22	AE	91	0046B	55\$:	CMPB	RESULT_DESC+2, #33	1300
					06	13	0046F		BEQL	56\$	
			20	22	AE	91	00471		CMPB	RESULT_DESC+2, #32	1301
					2E	12	00475		BNEQ	59\$	
24	BE		08		00	0C	00477	56\$:	PROBER	#0, #8, @RESULT_DESC+4	1304
					0E	12	0047C		BNEQ	57\$	
			24		AE	DD	0047E		PUSHL	RESULT_DESC+4	1306
					01	DD	00481		PUSHL	#1	
			00028228		8F	DD	00483		PUSHL	#164392	
					03	FB	00489		CALLS	#3, LIB\$SIGNAL	
		24	AE	24	BE	D0	0048C	57\$:	MOVL	@RESULT_DESC+4, RESULT_DESC+4	1307
			21	22	AE	91	00491		CMPB	RESULT_DESC+2, #33	1308
					04	12	00495		BNEQ	58\$	
		22	AE		16	90	00497		MOVB	#22, RESULT_DESC+2	1310
			20	22	AE	91	0049B	58\$:	CMPB	RESULT_DESC+2, #32	1311
					04	12	0049F		BNEQ	59\$	
		22	AE		17	90	004A1		MOVB	#23, RESULT_DESC+2	1313
		24	AE	14	A6	C0	004A5	59\$:	ADDL2	20(DATA SUBNODE), RESULT_DESC+4	1317
59			6A		01	E1	004AA		BBC	#1, (R10), 66\$	1319
57			6A		02	E1	004AE		BBC	#2, (R10), 67\$	1325
			50	10	A7	32	004B2		CVTBL	16(R7), R0	1328
		18	AE		50	C0	004B6		ADDL2	R0, BIT_OFFSET	
		1C	AE	12	A7	3C	004BA		MOVZBL	18(R7), BIT_LENGTH	1329
				20	AE	D4	004BF		CLRL	RESULT_DESC	1332
			07	18	AE	93	004C2		BITB	BIT_OFFSET, #7	1333
					78	12	004C6		BNEQ	70\$	
			50	1C	AE	D0	004C8		MOVL	BIT_LENGTH, R0	1334
			08		50	D1	004CC		CMPL	R0, #8	1337
					0E	12	004CF		BNEQ	61\$	
05			6A		03	E1	004D1		BBC	#3, (R10), 60\$	1338
			50		06	D0	004D5		MOVL	#6, R0	
					29	11	004D8		BRB	65\$	
			50		02	D0	004DA	60\$:	MOVL	#2, R0	
					24	11	004DD		BRB	65\$	
			10		50	D1	004DF	61\$:	CMPL	R0, #16	1340
					0E	12	004E2		BNEQ	63\$	
05			6A		03	E1	004E4		BBC	#3, (R10), 62\$	1341
			50		07	D0	004E8		MOVL	#7, R0	
					16	11	004EB		BRB	65\$	
			50		03	D0	004ED	62\$:	MOVL	#3, R0	
					11	11	004F0		BRB	65\$	
			20		50	D1	004F2	63\$:	CMPL	R0, #32	1343
					49	12	004F5		BNEQ	70\$	
05			6A		03	E1	004F7		BBC	#3, (R10), 64\$	1344
			50		08	D0	004FB		MOVL	#8, R0	
					03	11	004FE		BRB	65\$	
			50		04	D0	00500	64\$:	MOVL	#4, R0	
		22	AE		50	90	00503	65\$:	MOVB	R0, RESULT_DESC+2	
					37	11	00507	66\$:	BRB	70\$	
			25	22	AE	91	00509	67\$:	CMPB	RESULT_DESC+2, #37	1355
					0A	12	0050D		BNEQ	68\$	
		22	AE		8F	B0	0050F		MOVB	#270, RESULT_DESC+2	1359
		24	AE	010E	02	C0	00515		ADDL2	#2, RESULT_DESC+4	1360

		50	10	A7	32	00519	68%:	CVTWL	16(R7), R0	1362
		AE		50	C0	0051D		ADDL2	R0, RESULT_DESC+4	
1C	AE	50	12	A7	3C	00521		MOVZWL	18(R7), R0	1363
		50		03	78	00525		ASHL	#3, R0, BIT_LENGTH	
		0E	22	AE	91	0052A		CMPB	RESULT_DESC+2, #14	1373
				0D	1F	0052E		BLSSU	69%	
		14	22	AE	91	00530		CMPB	RESULT_DESC+2, #20	1374
				07	1A	00534		BGTRU	69%	
		20	AE	12	A7	B0	00536	MOVW	18(R7), RESULT_DESC	1376
				03	11	0053B		BRB	70%	
			20	AE	D4	0053D	69%:	CLRL	RESULT_DESC	1381
		07	18	AE	93	00540	70%:	BITB	BIT_OFFSET, #7	1390
				42	13	00544		BEQL	75%	
		23	AE		0D	90	00546	MOVB	#13, RESULT_DESC+3	1396
			22	AE	95	0054A		TSTB	RESULT_DESC+2	1397
				04	12	0054D		BNEQ	71%	
		22	AE	22	90	0054F		MOVB	#34, RESULT_DESC+2	1398
		50	22	AE	9A	00553	71%:	MOVZBL	RESULT_DESC+2, R0	1404
		22		50	91	00557		CMPB	R0, #34	
				14	13	0055A		BEQL	72%	
		01		50	91	0055C		CMPB	R0, #1	1405
				0F	13	0055F		BEQL	72%	
		2A		50	91	00561		CMPB	R0, #42	1406
				0A	13	00564		BEQL	72%	
		29		50	91	00566		CMPB	R0, #41	1407
				05	13	00569		BEQL	72%	
		28		50	91	0056B		CMPB	R0, #40	1408
				05	12	0056E		BNEQ	73%	
		50		01	D0	00570	72%:	MOVL	#1, R0	1404
				03	11	00573		BRB	74%	
		50		08	D0	00575	73%:	MOVL	#8, R0	
51	1C	AE		50	C7	00578	74%:	DIVL3	R0, BIT_LENGTH, R1	
	20	AE		51	B0	0057D		MOVW	R1, RESULT_DESC	
	28	AE	18	AE	D0	00581		MOVL	BIT_OFFSET, RESULT_DESC+8	1410
				3E	11	00586		BRB	78%	1390
50	18	AE		08	C7	00588	75%:	DIVL3	#8, BIT_OFFSET, R0	1419
	24	AE		50	C0	0058D		ADDL2	R0, RESULT_DESC+4	
			23	AE	95	00591		TSTB	RESULT_DESC+3	1420
				30	12	00594		BNEQ	78%	
		50	1C	AE	D0	00596		MOVL	BIT_LENGTH, R0	1423
		07		50	93	0059A		BITB	R0, #7	
				0A	12	0059D		BNEQ	76%	
51		50		08	C7	0059F		DIVL3	#8, R0, R1	1429
	20	AE		51	B0	005A3		MOVW	R1, RESULT_DESC	
				1D	11	005A7		BRB	78%	
	22	AE		01	90	005A9	76%:	MOVB	#1, RESULT_DESC+2	1436
	00010000	BF		50	D1	005AD		CMPL	R0, #65536	1437
				09	1E	005B4		BGEQU	77%	
	20	AE		50	B0	005B6		MOVW	R0, RESULT_DESC	1440
			28	AE	D4	005BA		CLRL	RESULT_DESC+8	1441
				07	11	005BD		BRB	78%	1437
			20	AE	B4	005BF	77%:	CLRW	RESULT_DESC	1449
	28	AE		50	D0	005C2		MOVL	R0, RESULT_DESC+8	1450
08	BC	AE		0C	28	005C6	78%:	MOVC3	#12, RESULT_DESC, @VMS_DESC	1456
	20	50		01	D0	005CC		MOVL	#1, R0	1458
				04	005CF			RET		1460

DBGVALUES
V04-000

D 14
16-Sep-1984 02:45:26
14-Sep-1984 12:17:54

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGVALUES.B32;1

Page 48
(17)

; Routine Size: 1488 bytes, Routine Base: DBG\$CODE + 0546

```
1345 1461 1 GLOBAL ROUTINE DBG$PRIM_TO_VAL (prm_desc,target_type,val_desc) =
1346 1462 1 ++
1347 1463 1 FUNCTIONAL DESCRIPTION:
1348 1464 1
1349 1465 1 Translates language independent descriptors to language independent
1350 1466 1 value descriptors. Normally the input descriptor will be a primary
1351 1467 1 descriptor (a symbolic address reference), but it is also possible
1352 1468 1 to call this routine passing it a 'volatile value descriptor' (used
1353 1469 1 for absolute addressing) or a normal value descriptor (indirection).
1354 1470 1 This routine should be able to use the symbol table access routines
1355 1471 1 and the information contained within the primary descriptor to make
1356 1472 1 a descriptor representing a 'value materialization' for the object
1357 1473 1 described by the input descriptor.
1358 1474 1
1359 1475 1 Note that this routine must be able to use life-time, invocation, and
1360 1476 1 generation information to produce an accurate value descriptor of the
1361 1477 1 input object, or to decide when the value of an object cannot be
1362 1478 1 materialized (such as when the user's PC is not within the scope of
1363 1479 1 a dynamic variable).
1364 1480 1
1365 1481 1 Value descriptors produced by this routine must be marked (within the
1366 1482 1 type field of the language independent header block) as to whether
1367 1483 1 they are non-volatile (dbg$sk_value_desc) or volatile (dbg$sk_v_value_desc).
1368 1484 1 Volatile value descriptors will NOT be stored to represent '\', 'last value'.
1369 1485 1
1370 1486 1 Since value descriptors may be used as target descriptors ( as input to
1371 1487 1 dbg$ncob_type_conv ), some provision must be made for incorporating
1372 1488 1 a value pointer field within the value descriptor. This type of value
1373 1489 1 descriptor is loosely defined as a volatile type.
1374 1490 1
1375 1491 1 FORMAL PARAMETERS:
1376 1492 1
1377 1493 1 prm_desc - A longword containing the address of a primary descriptor
1378 1494 1
1379 1495 1 target_type - A longword containing the type of value descriptor
1380 1496 1 (dbg$sk_value_desc or dbg$sk_v_value_desc).
1381 1497 1
1382 1498 1 val_desc - The address of a longword to contain the address of the
1383 1499 1 resulting value descriptor
1384 1500 1
```

1386	1501	1	IMPLICIT INPUTS:
1387	1502	1	
1388	1503	1	NONE
1389	1504	1	
1390	1505	1	IMPLICIT OUTPUTS:
1391	1506	1	
1392	1507	1	In case of a success return, the resulting value descriptor must be
1393	1508	1	constructed from dynamic storage and returned.
1394	1509	1	
1395	1510	1	ROUTINE VALUE:
1396	1511	1	
1397	1512	1	An unsigned integer longword completion code
1398	1513	1	
1399	1514	1	COMPLETION CODES:
1400	1515	1	
1401	1516	1	STSSK_SUCCESS (1) - Success. Value descriptor constructed and returned.
1402	1517	1	
1403	1518	1	STSSK_ERROR (2) - Failure. Data-type is not known to DEBUG kernel.
1404	1519	1	
1405	1520	1	SIDE EFFECTS:
1406	1521	1	
1407	1522	1	In case of a severe error, this routine will SIGNAL the error.
1408	1523	1	
1409	1524	1	--


```
1411 1525 1 !DBG$PRIM_TO_VAL(prm_desc,val_desc,target_type)
1412 1526 2 BEGIN
1413 1527 3
1414 1528 4 Describe formal parameters that are structures
1415 1529 5
1416 1530 6 MAP
1417 1531 7     prm_desc      : REF dbg$primary;
1418 1532 8
1419 1533 9
1420 1534 10 Declare routine-level local variables
1421 1535 11
1422 1536 12 LOCAL
1423 1537 13     result_desc   : REF dbg$valdesc;
1424 1538 14     data_desc     : BLOCK [dbg$base_size+1, LONG]
1425 1539 15                     FIELD(dbg$dhdr_fields, dbg$value_fields);
1426 1540 16
1427 1541 17 BIND vms_desc = data_desc[dbg$a_value_vmsdesc] : dbg$stg_desc;
1428 1542 18
1429 1543 19 dbg$gl_current_primary = .prm_desc;
1430 1544 20
1431 1545 21 ! It is illegal to call DBG$PRIM_TO_VAL with a type.
1432 1546 22
1433 1547 23 IF .prm_desc[dbg$b_dhdr_kind] EQL rst$tk_type
1434 1548 24 THEN
1435 1549 25     BEGIN
1436 1550 26         LOCAL
1437 1551 27             name;
1438 1552 28             IF .prm_desc[dbg$l_dhdr_symid0] NEQ 0
1439 1553 29             THEN
1440 1554 30                 BEGIN
1441 1555 31                     dbg$sta_symname(.prm_desc[dbg$l_dhdr_symid0], name);
1442 1556 32                     SIGNAL (dbg$_novaltyp, 1, .name);
1443 1557 33                 END
1444 1558 34             ELSE
1445 1559 35                 SIGNAL (dbg$_novalue);
1446 1560 36             END;
1447 1561 37
1448 1562 38
1449 1563 39 ! First construct a VAX/VMS descriptor that for the desired value
1450 1564 40
1451 1565 41 IF .prm_desc[dbg$l_dhdr_symid0] NEQ 0 THEN dbg$sta_setcontext(.prm_desc[dbg$l_dhdr_symid0]);
1452 1566 42 dbg$collect(.prm_desc);
1453 1567 43
1454 1568 44 SELECT ONE .prm_desc[dbg$b_dhdr_type] OF
1455 1569 45     SET
1456 1570 46     [dbg$tk_primary_desc]:
1457 1571 47         IF NOT dbg$make_vms_desc(.prm_desc,vms_desc) THEN RETURN sts$tk_error;
1458 1572 48     [dbg$tk_v_value_desc]:
1459 1573 49         ch$move(12, prm_desc[dbg$a_value_vmsdesc], vms_desc);
1460 1574 50     [dbg$tk_value_desc]:
1461 1575 51         BEGIN
1462 1576 52             ch$fill(0, (dbg$tk_valdesc_base_size+1)*XUPVAL, data_desc);
1463 1577 53             data_desc[dbg$b_dhdr_lang] = .prm_desc[dbg$b_dhdr_lang];
1464 1578 54             data_desc[dbg$b_dhdr_type] = .prm_desc[dbg$b_dhdr_type];
1465 1579 55             data_desc[dbg$l_dhdr_symid0] = .prm_desc[dbg$l_dhdr_symid0];
1466 1580 56             data_desc[dbg$w_dhdr_length] = dbg$tk_valdesc_base_size*XUPVAL;
1467 1581 57             data_desc[dbg$b_dhdr_kind] = rst$tk_data;
```

```
1468 1582 3
1469 1583 3
1470 1584 3
1471 1585 3
1472 1586 3
1473 1587 3
1474 1588 3
1475 1589 3
1476 1590 4
1477 1591 4
1478 1592 4
1479 1593 4
1480 1594 4
1481 1595 4
1482 1596 4
1483 1597 4
1484 1598 4
1485 1599 4
1486 1600 4
1487 1601 4
1488 1602 4
1489 1603 4
1490 1604 4
1491 1605 3
1492 1606 2
1493 1607 2
1494 1608 2
1495 1609 2
1496 1610 2
1497 1611 2
1498 1612 2
1499 1613 2
1500 1614 2
1501 1615 2
1502 1616 2
1503 1617 2
1504 1618 2
1505 1619 2
1506 1620 2
1507 1621 2
1508 1622 2
1509 1623 2
1510 1624 2
1511 1625 2
1512 1626 2
1513 1627 2
1514 1628 2
1515 1629 2
1516 1630 2
1517 1631 2
1518 1632 2
1519 1633 3
1520 1634 4
1521 1635 4
1522 1636 4
1523 1637 4
1524 1638 3

data_desc[dbg$b_dhdr_fcode] = rst$k_type_descr;

!+
!- Passing a pointer value into DBG$PRIM_TO_VAL results in a
!- value which is obtained by dereferencing the pointer.
IF .prm_desc[dbg$b_dhdr_fcode] EQL rst$k_type_tptr
THEN
BEGIN
LOCAL bit_length, bit_offset;
dbg$sta_type_typedptr(.prm_desc[dbg$l_dhdr_typeid], data_desc[dbg$l_dhdr_typeid]);
data_desc[dbg$b_dhdr_fcode] = dbg$sta_type_fcode(data_desc[dbg$l_dhdr_typeid]);
IF .data_desc[dbg$b_dhdr_fcode] EQL rst$k_type_array THEN RETURN 0;
bit_length = 0;
bit_offset = 0;
vms_desc[dsc$b_class] = 0;
vms_desc[dsc$b_dtype] = 0;
vms_desc[dsc$b_length] = 0;
vms_desc[dsc$a_pointer] = (.prm_desc[dbg$l_value_pointer]);
dbg$fill_in_vms_desc(.data_desc[dbg$b_dhdr_fcode],
                     .data_desc[dbg$l_dhdr_typeid],
                     .prm_desc[dbg$l_dhdr_symid0],
                     vms_desc, bit_length, bit_offset);
END;
END;
[OTHERWISE]:
SIGNAL(dbg$_filltype);
TES;

result_desc = dbg$make_val_desc(vms_desc, .target_type);

result_desc[dbg$b_dhdr_lang] = .prm_desc[dbg$b_dhdr_lang];
result_desc[dbg$b_dhdr_kind] = .prm_desc[dbg$b_dhdr_kind];
result_desc[dbg$b_dhdr_fcode] = .prm_desc[dbg$b_dhdr_fcode];
result_desc[dbg$l_dhdr_typeid] = .prm_desc[dbg$l_dhdr_typeid];
result_desc[dbg$l_dhdr_symid0] = .prm_desc[dbg$l_dhdr_symid0];
result_desc[dbg$v_dhdr_override] = .prm_desc[dbg$v_dhdr_override];

!+
!- Special case in COBOL. Treat the cobol record as text string.
!-
IF .result_desc[dbg$b_dhdr_fcode] EQL rst$k_type_record
THEN
BEGIN
LOCAL tmp_symid: REF rst$entry;
tmp_symid = .result_desc[dbg$l_dhdr_symid0];
WHILE .tmp_symid[rst$b_kind] NEQ rst$k_module DO
tmp_symid = .tmp_symid[rst$l_upscopeptr];
IF .tmp_symid[rst$b_language] EQL dbg$k_cobol
THEN
BEGIN
result_desc[dbg$b_dhdr_fcode] = rst$k_type_descr;
result_desc[dbg$b_value_class] = dsc$k_class_s;
result_desc[dbg$b_value_dtype] = dsc$k_dtype_t;
END;
```

```
1525 1639 2
1526 1640
1527 1641
1528 1642
1529 1643
1530 1644
1531 1645
1532 1646
1533 1647
1534 1648
1535 1649
1536 1650
1537 1651
1538 1652
1539 1653
1540 1654
1541 1655
1542 1656
1543 1657
1544 1658
1545 1659
1546 1660
1547 1661
1548 1662
1549 1663
1550 1664
1551 1665
1552 1666 1
```

```
END;

+
Special case for subrange - if we are turning a subrange primary into
a value descriptor, then change the type to reflect the parent type.
This is because, in all operations, a subrange is treated the same
as its parent type. This thus simplifies the operator tables.
-
If .target_type EQL dbg$sk_value_desc THEN
  WHILE .result_desc[dbg$b_dhdr_fcode] EQL rst$sk_type_subrng DO
    BEGIN
      LOCAL parent,fcode,typeid,bit_length,bit_offset,dummy;
      dbg$sta_typ_subrng(.result_desc[dbg$l_dhdr_typeid],parent,dummy,dummy,bit_length);
      bit_offset = 0;
      dbg$sta_syntype(.parent,fcode,typeid);
      result_desc[dbg$b_dhdr_fcode] = .fcode;
      result_desc[dbg$l_dhdr_typeid] = .typeid;
      result_desc[dbg$b_value_class] = 0;
      result_desc[dbg$b_value_dtype] = 0;
      result_desc[dbg$b_value_length] = 0;
      dbg$fill_in_vms_desc(.fcode,.typeid,.prm_desc[dbg$l_dhdr_symid0],
                           result_desc[dbg$a_value_vmsdesc],bit_length,bit_offset);
    END;

    .val_desc = .result_desc;
  RETURN sts$sk_success;

END;                                ! End of dbg$prim_to_val
```

58	00000000G	00	9E	00000	.ENTRY	DBG\$PRIM TO_VAL, Save R2,R3,R4,R5,R6,R7,R8	1461
5E	B8	AE	9E	00002	MOVAB	LIB\$SIGNAL, R8	
56	04	AC	DO	0000D	MOVAB	-72(SP), SP	
00	00000000G	56	DO	00011	MOVL	PRM_DESC, R6	1543
57	04	A6	9E	00018	MOVL	R6, DBG\$GL_CURRENT_PRIMARY	
07	03	A7	91	0001C	MOVAB	4(R6), R7	1547
		29	12	00020	CMPB	3(R7), #7	
	0C	A6	D5	00022	BNEQ	2\$	
		1B	13	00025	TSTL	12(R6)	1552
		5E	DD	00027	BEQL	1\$	
	0C	A6	DD	00029	PUSHL	SP	1555
00	00000000G	02	FB	0002C	PUSHL	12(R6)	
		6E	DD	00033	CALLS	#2, DBG\$STA_SYMNAME	
		01	DD	00035	PUSHL	NAME	1556
68	00028168	8F	DD	00037	PUSHL	#1	
		03	FB	0003D	PUSHL	#164200	
		09	11	00040	CALLS	#3, LIB\$SIGNAL	
68	000287F8	8F	DD	00042	BRB	2\$	1552
		01	FB	00048	PUSHL	#165880	1559
	0C	A6	D5	0004B	CALLS	#1, LIB\$SIGNAL	
		0A	13	0004E	TSTL	12(R6)	1565
	0C	A6	DD	00050	BEQL	3\$	
00	00000000G	01	FB	00053	PUSHL	12(R6)	
					CALLS	#1, DBG\$STA_SETCONTEXT	

					56	DD	0005A	3%:	PUSHL	R6		1566
	00000000G	00			01	FB	0005C		CALLS	#1, DBG\$COLLECT		
	79	8F		02	A6	91	00063		CMPB	2(R6), #121		1570
					11	12	00068		BNEQ	4%		
				38	AE	9F	0006A		PUSHAB	VMS_DESC		1571
					56	DD	0006D		PUSHL	R6		
	F9BC	CF			02	FB	0006F		CALLS	#2, DBG\$MAKE_VMS_DESC		
		11			50	EB	00074		BLBS	R0, 5%		
		50			02	DD	00077		MOVL	#2, R0		
					04	0007A			RET			
	83	8F		02	A6	91	0007B	4%:	CMPB	2(R6), #131		1572
					08	12	00080		BNEQ	6%		
38	AE	14	A6		0C	28	00082		MOVC3	#12, 20(R6), VMS_DESC		1573
					7A	11	00088	5%:	BRB	9%		
	7A	8F		02	A6	91	0008A	6%:	CMPB	2(R6), #122		1574
					6A	12	0008F		BNEQ	8%		
24	00	6E			00	2C	00091		MOVC5	#0, (SP), #0, #36, DATA_DESC		1576
				24	AE		00096					
	26	AE		02	A6	80	00098		MOVW	2(R6), DATA_DESC+2		1578
	30	AE		0C	A6	DD	0009D		MOVL	12(R6), DATA_DESC+12		1579
	24	AE			20	80	000A2		MOVW	#32, DATA_DESC		1580
	2A	AE		0603	8F	80	000A6		MOVW	#1539, DATA_DESC+6		1582
		06		02	A7	91	000AC		CMPB	2(R7), #6		1588
					52	12	000B0		BNEQ	9%		
				2C	AE	9F	000B2		PUSHAB	DATA_DESC+8		1592
				08	A6	DD	000B5		PUSHL	8(R6)		
	00000000G	00			02	FB	000B8		CALLS	#2, DBG\$STA_TYP_TYPEDPTR		
				2C	AE	9F	000BF		PUSHAB	DATA_DESC+8		1593
	00000000G	00			01	FB	000C2		CALLS	#1, DBG\$STA_TYPEFCODE		
	2A	AE			50	90	000C9		MOVB	R0, DATA_DESC+6		
		01		2A	AE	91	000CD		CMPB	DATA_DESC+6, #1		1594
					03	12	000D1		BNEQ	7%		
				00E6	31	000D3			BRW	15%		
				04	AE	7C	000D6	7%:	CLRQ	BIT_OFFSET		1596
				38	AE	D4	000D9		CLRL	VMS_DESC		1599
	3C	AE			B6	DD	000DC		MOVL	224(R6), VMS_DESC+4		1600
				04	AE	9F	000E1		PUSHAB	BIT_OFFSET		1601
				0C	AE	9F	000E4		PUSHAB	BIT_LENGTH		
				40	AE	9F	000E7		PUSHAB	VMS_DESC		
				0C	A6	DD	000EA		PUSHL	12(R6)		1603
				3C	AE	DD	000ED		PUSHL	DATA_DESC+8		1602
		7E		3E	AE	9A	000F0		MOVZBL	DATA_DESC+6, -(SP)		1601
	F6E0	CF			06	FB	000F4		CALLS	#6, DBG\$FILL_IN_VMS_DESC		
					09	11	000F9		BRB	9%		1568
				000287D8	8F	DD	000FB	8%:	PUSHL	#165848		1608
		68			01	FB	00101		CALLS	#1, LIB\$SIGNAL		
				08	AC	DD	00104	9%:	PUSHL	TARGET_TYPE		1611
				3C	AE	9F	00107		PUSHAB	VMS_DESC		
	F51E	CF			02	FB	0010A		CALLS	#2, DBG\$MAKE_VAL_DESC		
		52			50	DD	0010F		MOVL	R0, RESULT_DESC		
	03	A2		03	A6	90	00112		MOVB	3(R6), 3(RESULT_DESC)		1613
		53		04	A2	9E	00117		MOVAB	4(RESULT_DESC), R3		1614
	02	A3		02	A7	80	0011B		MOVW	2(R7), 2(R3)		1615
	08	A2		08	A6	7D	00120		MOVQ	8(R6), 8(RESULT_DESC)		1616
50		01			07	EF	00125		EXTZV	#7, #1, (R7), R0		1618
63		07			50	F0	0012A		INSV	R0, #7, #1, (R3)		
		07		02	A3	91	0012F		CMPB	2(R3), #7		1624

			20	12	00133		BNEQ	12\$		
	50	0C	A2	D0	00135		MOVL	12(RESET_DESC), TMP_SYMD	1629	
	01	14	A0	91	00139	10\$:	CMPB	20(TMP_SYMD), #1	1630	
			06	13	0013D		BEQL	11\$		
	50	10	A0	D0	0013F		MOVL	16(TMP_SYMD), TMP_SYMD	1631	
			F4	11	00143		BRB	10\$		
	03	29	A0	91	00145	11\$:	CMPB	41(TMP_SYMD), #3	1632	
			0A	12	00149		BNEQ	12\$		
02	A3		03	90	0014B		MOVB	#3, 2(R3)	1635	
16	A2	010E	8F	B0	0014F		MOVW	#270, 22(RESET_DESC)	1637	
0000007A	8F	08	AC	D1	00155	12\$:	CMPB	TARGET_TYPE, #122	1647	
			55	12	0015D		BNEQ	14\$		
	09	02	A3	91	0015F	13\$:	CMPB	2(R3), #9	1648	
			4F	12	00163		BNEQ	14\$		
			20	AE	9F	00165	PUSHAB	BIT_LENGTH	1651	
			10	AE	9F	00168	PUSHAB	DUMMY		
			14	AE	9F	0016B	PUSHAB	DUMMY		
			1C	AE	9F	0016E	PUSHAB	PARENT		
00000000G	00	08	A2	DD	00171		PUSHL	8(RESET_DESC)		
			05	FB	00174		CALLS	#5, DBG\$STA_TYP_SUBRNG		
			1C	AE	D4	0017B	CLRL	BIT_OFFSET	1652	
			14	AE	9F	0017E	PUSHAB	TYPEID	1653	
			1C	AE	9F	00181	PUSHAB	FCODE		
			18	AE	DD	00184	PUSHL	PARENT		
00000000G	00		03	FB	00187		CALLS	#3, DBG\$STA_SYMTYPE		
02	A3	18	AE	90	0018E		MOVB	FCODE, 2(R3)	1654	
08	A2	14	AE	D0	00193		MOVL	TYPEID, 8(RESET_DESC)	1655	
			14	A2	D4	0019B	CLRL	20(RESET_DESC)	1658	
			1C	AE	9F	0019B	PUSHAB	BIT_OFFSET	1660	
			24	AE	9F	0019E	PUSHAB	BIT_LENGTH		
			14	A2	9F	001A1	PUSHAB	20(RESET_DESC)		
			0C	A6	DD	001A4	PUSHL	12(R6)		
			24	AE	DD	001A7	PUSHL	TYPEID		
			2C	AE	DD	001AA	PUSHL	FCODE		
F627	CF		06	FB	001AD		CALLS	#6, DBG\$FILL_IN_VMS_DESC		
			AB	11	001B2		BRB	13\$	1648	
0C	BC		52	D0	001B4	14\$:	MOVL	RESULT_DESC, @VAL_DESC	1663	
	50		01	D0	001B8		MOVL	#1, R0	1664	
				04	001BB		RET			
			50	D4	001BC	15\$:	CLRL	R0	1666	
				04	001BE		RET			

; Routine Size: 447 bytes, Routine Base: DBG\$CODE + 0B16

```
1554 1667 1 GLOBAL ROUTINE DBG$PRINT_AGGREGATE(prm_desc,radix) : NOVALUE =
1555 1668 BEGIN
1556 1669 MAP prm_desc : REF dbg$primary;
1557 1670 BUILTIN-REMQUE;
1558 1671 LOCAL
1559 1672 subnode : REF dbg$prim_node,
1560 1673 val_desc : REF dbg$val_desc,
1561 1674 symId,kind,fcode,typeid,dummy,mark_one,mark_two;
1562 1675
1563 1676 dbg$gl_current_primary = .prm_desc; ! A003
1564 1677
1565 1678 dbg$newline();
1566 1679 dbg$print_control(dbg$k_prtset_rlmargin,+4); ! Indent by +4
1567 1680 subnode[dbg$prim_node_blink] = true;
1568 1681 subnode[dbg$prim_node_eval] = true;
1569 1682 SELECTONE .subnode[dbg$b_pnode_fcode] OF
1570 1683 SET
1571 1684 [rst$k_type_array]:
1572 1685 BEGIN
1573 1686 LABEL cell;
1574 1687 LOCAL s_vector : REF dbg$prim_node_subs;
1575 1688
1576 1689 s_vector = subnode[dbg$a_pnarr_svector];
1577 1690
1578 1691 ! Check for the array being empty (i.e., if any of the
1579 1692 ! dimensions has zero or negative length). If this is the
1580 1693 ! case, indicate that this is an empty array,
1581 1694 ! and then clean up and return.
1582 1695
1583 1696 INCR i FROM 0 to .subnode[dbg$b_pnarr_dimcnt]-1 DO
1584 1697 BEGIN
1585 1698 IF .s_vector[i,dbg$l_pnsub_lbound] GTR
1586 1699 .s_vector[i,dbg$l_pnsub_ubound]
1587 1700 THEN
1588 1701 BEGIN
1589 1702 dbg$print(UPRIT (%ASCIC '[empty array]'));
1590 1703 dbg$newline();
1591 1704 subnode[dbg$prim_node_eval] = false;
1592 1705 dbg$print_control(dbg$k_prtset_rlmargin,-4);
1593 1706 RETURN;
1594 1707 END;
1595 1708 END;
1596 1709
1597 1710 mark_one = dbg$push_tempmem();
1598 1711 dbg$sta_syntype(.subnode[dbg$l_pnarr_celltype],fcode,typeid);
1599 1712 dbg$build_primary_subnode(.prm_desc,rst$k_data,0,.fcode,.typeid,0);
1600 1713 dbg$collect(.prm_desc);
1601 1714 WHILE NOT .dbg$gv_control[dbg$gv_control_stop] DO
1602 1715 cell: BEGIN
1603 1716 mark_two = dbg$push_tempmem();
1604 1717 dbg$print_identifier(.prm_desc,0);
1605 1718 IF .prm_desc[dbg$gv_dhdr_agg]
1606 1719 THEN dbg$print_aggregate(.prm_desc,.radix)
1607 1720 ELSE
1608 1721 BEGIN
1609 1722 dbg$print(UPRIT BYTE(%ASCIC '!AD!_'),1,UPRIT BYTE(':'));
1610 1723
```

```
1611 1724 5
1612 1725 5
1613 1726 5
1614 1727 5
1615 1728 5
1616 1729 5
1617 1730 5
1618 1731 5
1619 1732 5
1620 1733 5
1621 1734 5
1622 1735 5
1623 1736 4
1624 1737 4
1625 1738 4
1626 1739 5
1627 1740 5
1628 1741 6
1629 1742 6
1630 1743 5
1631 1744 5
1632 1745 5
1633 1746 6
1634 1747 6
1635 1748 7
1636 1749 6
1637 1750 7
1638 1751 6
1639 1752 6
1640 1753 6
1641 1754 6
1642 1755 6
1643 1756 6
1644 1757 5
1645 1758 6
1646 1759 6
1647 1760 6
1648 1761 6
1649 1762 6
1650 1763 6
1651 1764 6
1652 1765 7
1653 1766 6
1654 1767 7
1655 1768 6
1656 1769 6
1657 1770 6
1658 1771 6
1659 1772 6
1660 1773 6
1661 1774 6
1662 1775 5
1663 1776 6
1664 1777 6
1665 1778 6
1666 1779 6
1667 1780 6
```

```
! If you examine a label array in PLI then you see the
! instructions at the labels. Instructions must always
! be represented as volatile value descriptors (since
! if you copy the bits, then the operands may change).
IF .fcode EQL rst$sk_type_self_rel_lab
THEN
  dbg$prim_to_val(.prm_desc,dbg$sk_v_value_desc,val_desc)
ELSE
  dbg$prim_to_val(.prm_desc,dbg$sk_value_desc,val_desc);
  dbg$print_value(.val_desc,.radix,.dbg$gl_sign_flag);
  dbg$newline();
END;
dbg$pop_tempmem(.mark_two);
INCR dimension FROM 1 TO .subnode[dbg$b_pnarr_dimcnt] DO
  BEGIN
    LOCAL s,typeid: REF rst$entry;
    s = (IF .subnode[dbg$b_pnarr_column]
      THEN .dimension - 1
      ELSE .subnode[dbg$b_pnarr_dimcnt] - .dimension);
    typeid = .s_vector[.s,dbg$l_pnsub_typeid];
    IF .s_vector[.s,dbg$l_pnsub_svalue] GEQ (
      IF (.dbg$gb_language EQL dbg$sk_ada) AND
      (.typeid NEQ 0)
      THEN
        IF (.typeid[rst$b_fcode] EQL rst$sk_type_enum)
        THEN
          dbg$enum_val(.typeid,.s_vector[.s,dbg$l_pnsub_ubound])
        ELSE
          .s_vector[.s,dbg$l_pnsub_ubound]
        ELSE
          .s_vector[.s,dbg$l_pnsub_ubound])
      THEN
        BEGIN
          ! For arrays indexed by enumeration types in ADA, the lower bound field gives
          ! the position, which we need to translate into a value.
          typeid = .s_vector[.s,dbg$l_pnsub_typeid];
          IF (.dbg$gb_language EQL dbg$sk_ada) AND
          (.typeid NEQ 0)
          THEN
            IF (.typeid[rst$b_fcode] EQL rst$sk_type_enum)
            THEN
              s_vector[.s,dbg$l_pnsub_svalue] = dbg$enum_val(.typeid,.s_vector[.s,dbg$l_pnsub_ubound])
            ELSE
              s_vector[.s,dbg$l_pnsub_svalue] = .s_vector[.s,dbg$l_pnsub_lbound]
            ELSE
              s_vector[.s,dbg$l_pnsub_svalue] = .s_vector[.s,dbg$l_pnsub_lbound];
          END
        ELSE
          BEGIN
            LOCAL
              s_value;
            ! For arrays indexed by enumeration types in ADA, we need to use ENUM_SUCC to get
```

DBGVALUES
V04-000

N 14
16-Sep-1984 02:45:26 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:17:54 [DEBUG.SRC]DBGVALUES.B32;1

Page 58
(21)

```
: 1668      1781      6
: 1669      1782      6
: 1670      1783      6
: 1671      1784      6
: 1672      1785      6
: 1673      1786      7
: 1674      1787      6
: 1675      1788      7
: 1676      1789      6
: 1677      1790      6
: 1678      1791      6
: 1679      1792      6
: 1680      1793      6
: 1681      1794      6
: 1682      1795      6
: 1683      1796      5
: 1684      1797      4
: 1685      1798      4
: 1686      1799      3
: 1687      1800      3
: 1688      1801      3
: 1689      1802      2
```

```
! the successor subscript. In all other cases, we can just add one.
s_value = .s_vector[.s,dbg$l_pnsub_svalue];
typeid = .s_vector[.s,dbg$l_pnsub_typeid];
IF (.dbg$gb_language EQL dbg$sk_ada) AND
   (.typeid NEQ 0)
THEN
  IF (.typeid[rst$b_fcode] EQL rst$sk_type_enum)
  THEN
    s_vector[.s,dbg$l_pnsub_svalue] = dbg$enum_succ(.typeid, .s_value)
  ELSE
    s_vector[.s,dbg$l_pnsub_svalue] = .s_vector[.s,dbg$l_pnsub_svalue] + 1
  ELSE
    s_vector[.s,dbg$l_pnsub_svalue] = .s_vector[.s,dbg$l_pnsub_svalue] + 1;
  LEAVE cell;
END;
END;
EXITLOOP;
END; ! End of block 'cell'
REMQUE(.prm_desc[dbg$l_prim_blink],dummy);
dbg$pop_tempmem(.mark_one);
END;
```



```

[rst$k_type_record,rst$k_type_variant]:
BEGIN
LOCAL n_comps,s_vector : REF VECTOR [,LONG];
IF .subnode[dbg$b_pnode_fcode] EQL rst$k_type_record
THEN
  dbg$sta_typ_record(.subnode[dbg$l_pnode_typeid],n_comps,s_vector,dummy)
ELSE
  BEGIN
    n_comps = .subnode[dbg$w_pnvar_ncomps];
    s_vector = .subnode[dbg$l_pnvar_complst];
  END;
INCR component FROM 0 TO .n_comps-1 DO
  IF (symid = .s_vector[.component]) NEQ 0 THEN
    BEGIN
      IF .dbg$gv_control[dbg$gv_control_stop] THEN EXITLOOP;
      IF .prm_desc[dbg$w_dhdr_tmprf] THEN
        BEGIN
          prm_desc[dbg$w_dhdr_tmprf] = false;
          prm_desc[dbg$w_dhdr_subref] = false;
          prm_desc[dbg$w_prim_offset] = 0;
          prm_desc[dbg$w_prim_length] = 0;
        END;
      mark_one = dbg$push_tempmem();
      dbg$sta_symkind(.symid,kind);
      IF .kind EQL rst$k_variant
      THEN
        BEGIN
          LOCAL
            tagid,tag_val,
            tag_name      : REF VECTOR[,BYTE],
            variant        : REF rst$var_entry;
          MAP symid        : REF rst$entry;

          variant = 0;
          IF (tagid = .symid[rst$l_vartagptr]) NEQ 0 THEN
            BEGIN
              dbg$sta_symname(.tagid,tag_name);
              IF .tag_name[0] NEQ 0 THEN
                BEGIN
                  dbg$sta_syntype(.tagid,fcode,typeid);
                  dbg$build_primary_subnode(.prm_desc,rst$k_data,.tagid,.fcode,.typeid,0);
                  dbg$prim_to_val(.prm_desc,dbg$w_value_desc,val_desc);
                  tag_val = .val_desc[dbg$l_value_value0];
                  variant = dbg$sta_variant_select(.tag_val,.symid);
                  REMQUE(.prm_desc[dbg$l_prim_blink],dummy);
                END;
              END;
            END;
          IF .variant EQL 0
          THEN
            BEGIN
              dbg$print(UPRINT(ASCII '!AD'),52,UPRINT BYTE('[Variant Record omitted - null or illeg
              dbg$newline());
            END;
          ELSE
            BEGIN
              dbg$build_primary_subnode(.prm_desc,rst$k_variant,0,rst$k_type_variant,0,0);
              subnode = .prm_desc[dbg$l_prim_blink];
            END;
          END;
        END;
      END;
    END;
  END;
END;

```

```
1748 1860 6 subnode[dbg$l_pnvar_tagid] = .tagid;
1749 1861 6 subnode[dbg$w_pnvar_index] = 1;
1750 1862 6 subnode[dbg$v_pnvar_valid] = true;
1751 1863 6 subnode[dbg$w_pnvar_ncomps] = .variant[rst$l_var_compcnt];
1752 1864 6 subnode[dbg$l_pnvar_complst] = .variant[rst$a_var_complst];
1753 1865 6 subnode[dbg$l_pnvar_dstptr] = .variant[rst$l_var_dstptr];
1754 1866 6 dbg$print(UPLIT(%ASCII '!AD'),30,UPLIT BYTE('Variant Record with Tag Value '));
1755 1867 6 dbg$print_value(.val_desc,.radix,.dbg$gl_sign_flag);
1756 1868 6 dbg$print_aggregate(.prm_desc,.radix);
1757 1869 6 REMQUE(.prm_desc[dbg$l_prim_blink],dummy);
1758 1870 5 END;
1759 1871 5 END
1760 1872 4 ELSE
1761 1873 5 BEGIN
1762 1874 5 dbg$sta_syntype(.symid,fcode,typeid);
1763 1875 5 dbg$buil_d_primary_subnode(.prm_desc,.kind,.symid,.fcode,.typeid,0);
1764 1876 5 dbg$collect(.prm_desc);
1765 1877 5 IF .prm_desc[dbg$v_dhdr_aggr]
1766 1878 5 THEN
1767 1879 6 BEGIN
1768 1880 6 dbg$print_identifier(.prm_desc,0);
1769 1881 6 dbg$print_aggregate(.prm_desc,.radix);
1770 1882 6 END
1771 1883 5 ELSE
1772 1884 6 BEGIN
1773 1885 6 LOCAL name : REF VECTOR[BYTE];
1774 1886 6 dbg$sta_symname(.symid,name);
1775 1887 6 IF .name[0] NEQ 0 THEN
1776 1888 7 BEGIN
1777 1889 7 dbg$print_identifier(.prm_desc,0);
1778 1890 7 dbg$print(UPLIT BYTE(%ASCII '!AD! '),1,UPLIT BYTE(':'));
1779 1891 7 dbg$prim_to_val(.prm_desc,dbg$sk_value_desc,val_desc);
1780 1892 7 dbg$print_value(.val_desc,.radix,.dbg$gl_sign_flag);
1781 1893 7 dbg$newline();
1782 1894 6 END;
1783 1895 5 END;
1784 1896 5 REMQUE(.prm_desc[dbg$l_prim_blink],dummy);
1785 1897 4 END;
1786 1898 4 dbg$pop_tempmem(.mark_one);
1787 1899 3 END;
1788 1900 2 END;
1789 1901 2 [OTHERWISE]:
1790 1902 2 SIGNAL(dbg$_illtype);
1791 1903 2
1792 1904 2
1793 1905 2 TES;
1794 1906 2
1795 1907 2 subnode[dbg$v_pnode_eval] = false;
1796 1908 2 prm_desc[dbg$v_dhdr_aggr] = true;
1797 1909 2 prm_desc[dbg$b_dhdr_kind] = .subnode[dbg$b_pnode_kind];
1798 1910 2 prm_desc[dbg$b_dhdr_fcode] = .subnode[dbg$b_pnode_fcode];
1799 1911 2 prm_desc[dbg$l_dhdr_typeid] = .subnode[dbg$l_pnode_typeid];
1800 1912 2 dbg$print_control(dbg$sk_prtset_rlmargin,-4); ! Reset indentation
1801 1913 1 END; ! End of dbg$print_aggregate
```

```
.PSECT DBG$PLIT, NOWRT, SHR, PIC, 0

00 5D 79 61 72 72 61 20 79 74 70 6D 65 5B 0D 00008 P.AAC: .ASCII <13>\[empty array]\<0><0>
      00 00017
      5F 21 44 41 21 05 00018 P.AAD: .ASCII <5>\!AD!\
      3A 0001E P.AAE: .ASCII \:\
      0001F .BLKB 1
64 72 6F 63 65 52 20 74 6E 61 69 44 41 21 03 00020 P.AAF: .ASCII <3>\!AD\
6C 6C 75 6E 20 2D 20 64 65 61 69 72 61 56 5B 00024 P.AAG: .ASCII \[Variant Record omitted - null or illegal\
      6C 6C 75 6E 20 2D 20 64 65 61 69 72 61 56 5B 00033
      5D 65 75 6C 61 56 20 67 61 54 20 6C 00042
      20 64 72 6F 63 65 52 20 74 6E 61 69 72 61 56 00058 P.AAH: .ASCII \[ Tag Value]\
      2D 65 75 6C 61 56 20 67 61 54 20 68 74 69 77 0005C P.AAI: .ASCII \Variant Record with Tag Value \
      5F 21 44 41 21 05 0007A P.AAJ: .ASCII <5>\!AD!\
      3A 00080 P.AAK: .ASCII \:\

.PSECT DBG$CODE, NOWRT, SHR, PIC, 0

      OFFC 00000
      5E 28 C2 00002
      58 04 AC D0 00005
00000000G 00 58 D0 00009
00000000G 00 00 FB 00010
      04 DD 00017
      02 DD 00019
00000000G 00 02 FB 0001B
      52 18 A8 D0 00022
      0A A2 01 88 00026
      50 09 A2 9A 0002A
      01 50 91 0002E
      03 13 00031
      01C2 31 00033
      53 28 A2 9E 00036 1$:
      55 18 A2 9A 0003A
      54 01 CE 0003E
      2C 11 00041
      50 54 14 C5 00043 2$:
      0C A043 9F 00047
      08 A043 9F 0004B
      9E 9E D1 0004F
      1B 15 00052
      00000000' EF 9F 00054
00000000G 00 01 FB 0005A
00000000G 00 00 FB 00061
      0A A2 01 8A 00068
      D3C5 31 0006C
      54 55 F2 0006F 3$:
      00 00 FB 00073
      6E 50 D0 0007A
      18 AE 9F 0007D
      20 AE 9F 00080
      24 A2 DD 00083

.SUBL2
MOV L DBGS$PRINT, AGGREGATE, Save R2,R3,R4,R5,R6,-
R7,R8,R9,R10,R11
#40, SP
PRM_DESC, R8
R8, DBGS$GL_CURRENT_PRIMARY
CALLS #0, DBGS$NEWLINE
PUSH L #4
PUSH L #2
CALLS #2, DBGS$PRINT CONTROL
MOV L 24(R8), SUBNODE
BISB2 #1, 10(SUBNODE)
MOVZBL 9(SUBNODE), R0
CMPB R0, #1
BEQ L 1$
BRW 23$
MOVAB 40(R2), S_VECTOR
MOVZBL 27(SUBNODE), R5
MNEGL #1, I
BRB 3$
MULL3 #20, I, R0
PUSHAB 12(R0)[S_VECTOR]
PUSHAB 8(R0)[S_VECTOR]
CML PL @ (SP)+, -@ (SP)+
BLEQ 3$
PUSHAB P.AAC
CALLS #1, DBGS$PRINT
CALLS #0, DBGS$NEWLINE
BICB2 #1, 10(SUBNODE)
BRW 42$
AOBLSS R5, I, 2$
CALLS #0, DBGS$PUSH_TEMPMEM
MOV L R0, MARK_ONE
PUSHAB TYPEID
PUSHAB FCODE
PUSH L 36(SUBNODE)
```

00000000G	00		03	FB	00086	CALLS	#3, DBG\$STA_SYMTYPE	
			7E	D4	0008D	CLRL	-(SP)	1712
		1C	AE	DD	0008F	PUSHL	TYPEID	
		24	AE	DD	00092	PUSHL	FCODE	
	7E		06	7D	00095	MOVQ	#6, -(SP)	
00000000G	00		58	DD	00098	PUSHL	R8	
			06	FB	0009A	CALLS	#6, DBG\$BUILD_PRIMARY_SUBNODE	
00000000G	00		58	DD	000A1	PUSHL	R8	1713
03 00000000G	00		01	FB	000A3	CALLS	#1, DBG\$COLLECT	
00000000G	00		01	E1	000AA	BBC	#1, DBG\$GV_CONTROL+1, 5\$	1714
			01	32	31	BRW	21\$	
00000000G	00		00	FB	000B5	CALLS	#0, DBG\$PUSH_TEMPMEM	1716
	5B		50	DD	000BC	MOVL	R0, MARK_TWO	
			7E	D4	000BF	CLRL	-(SP)	1717
00000000G	00		58	DD	000C1	PUSHL	R8	
	0C		02	FB	000C3	CALLS	#2, DBG\$PRINT_IDENTIFIER	
		04	A8	E9	000CA	BLBC	4(R8), 6\$	1718
		08	AC	DD	000CE	PUSHL	RADIX	1719
			58	DD	000D1	PUSHL	R8	
FF28	CF		02	FB	000D3	CALLS	#2, DBG\$PRINT_AGGREGATE	
			4A	11	000D8	BRB	9\$	
	00000000'		EF	9F	000DA	PUSHAB	P.AAE	1722
			01	DD	000E0	PUSHL	#1	
	00000000'		EF	9F	000E2	PUSHAB	P.AAD	
00000000G	00		03	FB	000E8	CALLS	#3, DBG\$PRINT	
	15	1C	AE	D1	000EF	CMPL	FCODE, #21	1729
			09	12	000F3	BNEQ	7\$	
		24	AE	9F	000F5	PUSHAB	VAL_DESC	1731
	7E	B3	8F	9A	000F8	MOVZBL	#13T, -(SP)	
			07	11	000FC	BRB	8\$	
		24	AE	9F	000FE	PUSHAB	VAL_DESC	1733
	7E	7A	8F	9A	00101	MOVZBL	#122, -(SP)	
			58	DD	00105	PUSHL	R8	
FD35	CF		03	FB	00107	CALLS	#3, DBG\$PRIM TO VAL	
		00000000G	00	DD	0010C	PUSHL	DBG\$GL_SIGN_FLAG	1734
			08	AC	DD	00112	PUSHL	RADIX
		2C	AE	DD	00115	PUSHL	VAL_DESC	
0000V	CF		03	FB	00118	CALLS	#3, DBG\$PRINT VALUE	
00000000G	00		00	FB	0011D	CALLS	#0, DBG\$NEWLINE	1735
			5B	DD	00124	PUSHL	MARK TWO	1737
00000000G	00		01	FB	00126	CALLS	#1, DBG\$POP_TEMPMEM	
	5A	1B	A2	9A	0012D	MOVZBL	27(SUBNODE), R10	1738
			54	D4	00131	CLRL	DIMENSION	
			7C	11	00133	BRB	16\$	
06	0A	A2	01	E1	00135	BBC	#1, 10(SUBNODE), 11\$	1741
		50	A4	9E	0013A	MOVAB	-1(R4), S	1742
			04	11	0013E	BRB	12\$	
50	5A		54	C3	00140	SUBL3	DIMENSION, R10, S	1743
56	50		14	C5	00144	MULL3	#20, S, R6	1745
		10	A643	9F	00148	PUSHAB	16(R6)[S VECTOR]	
			9E	DD	0014C	MOVL	2(SP)+, R9	
	59		59	DD	0014F	MOVL	R9, TYPEID	
	55		56	C1	00152	ADDL3	R6, S VECTOR, R7	1746
57	53		56	C1	00152	ADDL3	R6, S VECTOR, R7	
	50	0C	A643	9E	00156	MOVAB	12(R6)[S VECTOR], R0	1752
			00	91	0015B	CMPB	DBG\$GB_LANGUAGE, #9	1747
	09	00000000G	17	12	00162	BNEQ	13\$	
			55	D5	00164	TSTL	TYPEID	1748

			13	13	00166	BEQL	13\$		
	04	18	A5	91	00168	CMPB	24(TYPEID), #4		1750
			00	12	0016C	BNEQ	13\$		
			60	DD	0016E	PUSHL	(R0)		1752
00000000G	00		55	DD	00170	PUSHL	TYPEID		
			02	FB	00172	CALLS	#2, DBG\$ENUM_VAL		
			03	11	00179	BRB	14\$		
	50		60	DD	0017B	13\$: MOVL	(R0), R0		1756
	50		67	D1	0017E	14\$: CMPL	(R7), R0		1746
			30	19	00181	BLSS	17\$		
55			59	DD	00183	MOVL	R9, TYPEID		1763
50	08	A6	43	9E	00186	MOVAB	8(R6)[S_VECTOR], R0		1769
09	00000000G		00	91	0018B	CMPB	DBG\$GB_LANGUAGE, #9		1764
			1A	12	00192	BNEQ	15\$		
			55	D5	00194	TSTL	TYPEID		1765
			16	13	00196	BEQL	15\$		
04	18		A5	91	00198	CMPB	24(TYPEID), #4		1767
			10	12	0019C	BNEQ	15\$		
			60	DD	0019E	PUSHL	(R0)		1769
			55	DD	001A0	PUSHL	TYPEID		
00000000G	00		02	FB	001A2	CALLS	#2, DBG\$ENUM_VAL		
	67		50	DD	001A9	MOVL	R0, (R7)		
			33	11	001AC	BRB	20\$		
	67		60	DD	001AE	15\$: MOVL	(R0), (R7)		1773
			2E	11	001B1	16\$: BRB	20\$		1746
	50		67	DD	001B3	17\$: MOVL	(R7), S_VALUE		1783
	55		59	DD	001B6	MOVL	R9, TYPEID		1784
09	00000000G		00	91	001B9	CMPB	DBG\$GB_LANGUAGE, #9		1785
			1A	12	001C0	BNEQ	18\$		
			55	D5	001C2	TSTL	TYPEID		1786
			16	13	001C4	BEQL	18\$		
04	18		A5	91	001C6	CMPB	24(TYPEID), #4		1788
			10	12	001CA	BNEQ	18\$		
			50	DD	001CC	PUSHL	S_VALUE		1790
			55	DD	001CE	PUSHL	TYPEID		
00000000G	00		02	FB	001D0	CALLS	#2, DBG\$ENUM_SUCC		
	67		50	DD	001D7	MOVL	R0, (R7)		
			02	11	001DA	BRB	19\$		
			67	D6	001DC	18\$: INCL	(R7)		1794
			FEC9	31	001DE	19\$: BRW	4\$		1795
FF4E	54	01	5A	F1	001E1	20\$: ACBL	R10, #1, DIMENSION, 10\$		1738
	04	AE	B8	0F	001E7	21\$: REMQUE	24(R8), DUMMY		1800
			6E	DD	001EC	PUSHL	MARK ONE		1801
00000000G	00		01	FB	001EE	CALLS	#1, DBG\$POP_TEMPHEM		
			0221	31	001F5	22\$: BRW	41\$		1682
	07		50	91	001F8	23\$: CMPB	R0, #7		1803
			08	13	001FB	BEQL	24\$		
	13		50	91	001FD	CMPB	R0, #19		
			03	13	00200	BEQL	24\$		
			0207	31	00202	BRW	40\$		
	07		50	91	00205	24\$: CMPB	R0, #7		1806
			15	12	00208	BNEQ	25\$		
		04	AE	9F	0020A	PUSHAB	DUMMY		1808
		0C	AE	9F	0020D	PUSHAB	S_VECTOR		
		14	AE	9F	00210	PUSHAB	N_COMPS		
		0C	A2	DD	00213	PUSHL	12(SUBNODE)		
00000000G	00		04	FB	00216	CALLS	#4, DBG\$STA_TYP_RECORD		

			0A	11	0021D		BRB	26\$		
	0C	AE	1A	A2	3C	0021F	25\$:	MOVZWL	26(SUBNODE), N_COMPS	1811
	08	AE	20	A2	D0	00224		MOVL	32(SUBNODE), S_VECTOR	1812
		53	0C	AE	D0	00229	26\$:	MOVL	N_COMPS, R3	1814
		54		01	CE	0022D		MNEGL	#T, COMPONENT	
				01D0	31	00230	27\$:	BRW	38\$	
		57	08	BE44	D0	00233	28\$:	MOVL	2S VECTOR[COMPONENT], SYMID	1815
				F6	13	00238		BEQL	27\$	
B3	00000000G	00		01	E0	0023A		BBS	#1, DBG\$GV CONTROL+1, 22\$	1817
		55	04	AC	D0	00242		MOVL	PRM_DESC, R5	1818
		09	05	A5	E9	00246		BLBC	5(R5), 29\$	
	04	A5	0102	8F	AA	0024A		BICW2	#258, 4(R5)	1820
			10	A5	D4	00250		CLRL	16(R5)	1822
	00000000G	00		00	FB	00253	29\$:	CALLS	#0, DBG\$PUSH_TEMPMEM	1825
		6E		50	D0	0025A		MOVL	R0, MARK_ONE	
			10	AE	9F	0025D		PUSHAB	KIND	1826
				57	DD	00260		PUSHL	SYMID	
	00000000G	00		02	FB	00262		CALLS	#2, DBG\$STA_SYMKIND	
		08	10	AE	D1	00269		CMPL	KIND, #11	1827
				03	13	0026D		BEQL	30\$	
				00E3	31	0026F		BRW	33\$	
				5A	D4	00272	30\$:	CLRL	VARIANT	1836
		56	10	A7	D0	00274		MOVL	16(SYMID), TAGID	1837
				5E	13	00278		BEQL	31\$	
			14	AE	9F	0027A		PUSHAB	TAG_NAME	1839
				56	DD	0027D		PUSHL	TAGID	
	00000000G	00		02	FB	0027F		CALLS	#2, DBG\$STA_SYMNAME	
			14	BE	95	00286		TSTB	@TAG_NAME	1840
				4D	13	00289		BEQL	31\$	
			18	AE	9F	0028B		PUSHAB	TYPEID	1842
			20	AE	9F	0028E		PUSHAB	FCODE	
				56	DD	00291		PUSHL	TAGID	
	00000000G	00		03	FB	00293		CALLS	#3, DBG\$STA_SYMTYPE	
				7E	D4	0029A		CLRL	-(SP)	1843
			1C	AE	DD	0029C		PUSHL	TYPEID	
			24	AE	DD	0029F		PUSHL	FCODE	
				56	DD	002A2		PUSHL	TAGID	
				06	DD	002A4		PUSHL	#6	
				55	DD	002A6		PUSHL	R5	
	00000000G	00		06	FB	002A8		CALLS	#6, DBG\$BUILD_PRIMARY_SUBNODE	
			24	AE	9F	002AF		PUSHAB	VAL_DESC	1844
		7E	7A	8F	9A	002B2		MOVZBL	#122, -(SP)	
				55	DD	002B6		PUSHL	R5	
	FB84	CF		03	FB	002B8		CALLS	#3, DBG\$PRIM_TO_VAL	
		50	24	AE	D0	002BD		MOVL	VAL_DESC, R0	1845
		50	20	A0	D0	002C1		MOVL	32(R0), TAG_VAL	
			0081	8F	BB	002C5		PUSHR	#*M<R0,R7>	1846
	00000000G	00		02	FB	002C9		CALLS	#2, DBG\$STA_VARIANT_SELECT	
		5A		50	D0	002D0		MOVL	R0, VARIANT	
	04	AE	18	B5	0F	002D3	31\$:	REMQUE	@24(R5), DUMMY	1847
				5A	D5	002D8		TSTL	VARIANT	1850
				1F	12	002DA		BNEQ	32\$	
		00000000'		EF	9F	002DC		PUSHAB	P_AAG	1853
				34	DD	002E2		PUSHL	#52	
		00000000'		EF	9F	002E4		PUSHAB	P_AAF	
	00000000G	00		03	FB	002EA		CALLS	#3, DBG\$PRINT	
	00000000G	00		00	FB	002F1		CALLS	#0, DBG\$NEWLINE	1854

			00FF	31	002FB	BRW	37\$	1850
			7E	7C	002FB	32\$: CLRQ	-(SP)	1858
			13	DD	002FD	PUSHL	#19	
	7E		0B	7D	002FF	MOVQ	#11, -(SP)	
	55	04	AC	DD	00302	MOVL	PRM_DESC, R5	
			55	DD	00306	PUSHL	R5	
00000000G	00		06	FB	00308	CALLS	#6, DBG\$BUILD_PRIMARY_SUBNODE	1859
	52	18	A5	DD	0030F	MOVL	24(R5), SUBNODE	1860
1C	A2		56	DD	00313	MOVL	TAGID, 28(SUBNODE)	1861
18	A2		01	BD	00317	MOVW	#1, 24(SUBNODE)	1862
0A	A2		10	88	0031B	BISB2	#16, 10(SUBNODE)	1863
1A	A2	04	AA	BD	0031F	MOVW	4(VARIANT), 26(SUBNODE)	1864
20	A2	08	AA	9E	00324	MOVAB	8(R10), 32(SUBNODE)	1865
24	A2		6A	DD	00329	MOVL	(VARIANT), 36(SUBNODE)	1866
		00000000'	EF	9F	0032D	PUSHAB	P.AAI	
			1E	DD	00333	PUSHL	#30	
		00000000'	EF	9F	00335	PUSHAB	P.AAH	
00000000G	00		03	FB	0033B	CALLS	#3, DBG\$PRINT	
		00000000G	0C	DD	00342	PUSHL	DBG\$GL_SIGN_FLAG	1867
			08	AC	DD	PUSHL	RADIX	
		2C	AE	DD	0034B	PUSHL	VAL_DESC	
0000V	CF		03	FB	0034E	CALLS	#3, DBG\$PRINT_VALUE	
			3D	11	00353	BRB	34\$	1868
		18	AE	9F	00355	33\$: PUSHAB	TYPEID	1874
		20	AE	9F	00358	PUSHAB	FCODE	
			57	DD	0035B	PUSHL	SYMID	
00000000G	00		03	FB	0035D	CALLS	#3, DBG\$STA_SYMTYPE	
			7E	D4	00364	CLRL	-(SP)	1875
		1C	AE	DD	00366	PUSHL	TYPEID	
		24	AE	DD	00369	PUSHL	FCODE	
			57	DD	0036C	PUSHL	SYMID	
		20	AE	DD	0036E	PUSHL	KIND	
			55	DD	00371	PUSHL	R5	
00000000G	00		06	FB	00373	CALLS	#6, DBG\$BUILD_PRIMARY_SUBNODE	1876
			55	DD	0037A	PUSHL	R5	
00000000G	00		01	FB	0037C	CALLS	#1, DBG\$COLLECT	
	17	04	A5	E9	00383	BLBC	4(R5), 35\$	1877
			7E	D4	00387	CLRL	-(SP)	1880
			55	DD	00389	PUSHL	R5	
00000000G	00		02	FB	0038B	CALLS	#2, DBG\$PRINT_IDENTIFIER	
		08	AC	DD	00392	34\$: PUSHL	RADIX	1881
			55	DD	00395	PUSHL	R5	
FC64	CF		02	FB	00397	CALLS	#2, DBG\$PRINT_AGGREGATE	
			57	11	0039C	BRB	36\$	1877
		20	AE	9F	0039E	35\$: PUSHAB	NAME	1886
			57	DD	003A1	PUSHL	SYMID	
00000000G	00		02	FB	003A3	CALLS	#2, DBG\$STA_SYMNAME	
		20	BE	95	003AA	TSTB	@NAME	1887
			46	13	003AD	BEQL	36\$	
			7E	D4	003AF	CLRL	-(SP)	1889
			55	DD	003B1	PUSHL	R5	
00000000G	00		02	FB	003B3	CALLS	#2, DBG\$PRINT_IDENTIFIER	
		00000000'	EF	9F	003BA	PUSHAB	P.AAK	1890
			01	DD	003C0	PUSHL	#1	
		00000000'	EF	9F	003C2	PUSHAB	P.AAJ	
00000000G	00		03	FB	003C8	CALLS	#3, DBG\$PRINT	
		24	AE	9F	003CF	PUSHAB	VAL_DESC	1891

	7E	7A	8F	9A	003D2	MOVZBL	#122, -(SP)	
			55	DD	003D6	PUSHL	R5	
FA64	CF		03	FB	003D8	CALLS	#3, DBG\$PRIM TO VAL	
		00000000G	00	DD	003DD	PUSHL	DBG\$GL_SIGN_FLAG	1892
		08	AC	DD	003E3	PUSHL	RADIX	
		2C	AE	DD	003E6	PUSHL	VAL_DESC	
0000V	CF		03	FB	003E9	CALLS	#3, DBG\$PRINT VALUE	1893
00000000G	00		00	FB	003EE	CALLS	#0, DBG\$NEWLINE	1896
04	AE	18	B5	OF	003F5	REMOVE	324(R5), DUMMY	1898
			6E	DD	003FA	PUSHL	MARK JNE	
00000000G	00		01	FB	003FC	CALLS	#1, DBG\$POP TEMPMEM	
02	54		53	F2	00403	A0BLSS	R3, COMPONENT, 398	1815
			10	11	00407	BRB	418	1682
			FE27	31	00409	BRW	288	1815
		000287D8	8F	DD	0040C	PUSHL	#165848	1903
00000000G	00		01	FB	00412	CALLS	#1, LIB\$SIGNAL	
0A	A2		01	8A	00419	BICB2	#1, 10(SUBNODE)	1907
	50	04	AC	D0	0041D	MOVL	PRM_DESC, R0	1908
04	A0		01	88	00421	BISB2	#1, 4(R0)	
07	A0	08	A2	90	00425	MOVB	8(SUBNODE), 7(R0)	1909
06	A0	09	A2	90	0042A	MOVB	9(SUBNODE), 6(R0)	1910
08	A0	0C	A2	D0	0042F	MOVL	12(SUBNODE), 8(R0)	1911
	7E		04	CE	00434	MNEGL	#4, -(SP)	1912
			02	DD	00437	PUSHL	#2	
00000000G	00		02	FB	00439	CALLS	#2, DBG\$PRINT_CONTROL	
			04	04	00440	RET		1913

; Routine Size: 1089 bytes, Routine Base: DBG\$CODE + 0CD5


```
1803 1914 1 GLOBAL ROUTINE DBG$PRINT_VALUE(val_desc: REF dbg$valdesc, radix) : NOVALUE =
1804 1915 BEGIN
1805 1916 BUILTIN ACTUALCOUNT, ACTUALPARAMETER;
1806 1917
1807 1918 LOCAL
1808 1919     sign_flag,
1809 1920     save_flag,
1810 1921     vms_desc      : dbg$stg_desc;
1811 1922
1812 1923 sign_flag = (actualcount() GTR 2 AND actualparameter(3));
1813 1924 save_flag = (actualcount() LSS 4 OR actualparameter(4));
1814 1925
1815 1926 IF .save_flag THEN dbg$save_val(.val_desc);
1816 1927 ch$move(T2, val_desc[dbg$a_value_vmsdesc], vms_desc);
1817 1928
1818 1929 IF .val_desc[dbg$vh_hdr_format] NEQ 0 THEN
1819 1930 BEGIN
1820 1931     SELECT ONE .val_desc[dbg$vh_hdr_format] OF
1821 1932     SET
1822 1933     [1]: BEGIN          ! Condition Value
1823 1934         LOCAL
1824 1935             msgbuffer  : VECTOR [256, BYTE],
1825 1936             msg_desc   : dbg$stg_desc;
1826 1937
1827 1938             msg_desc[dsc$b_class] = dsc$b_class_s;
1828 1939             msg_desc[dsc$b_dtype] = dsc$b_dtype_t;
1829 1940             msg_desc[dsc$b_length] = 256;
1830 1941             msg_desc[dsc$a_pointer] = msgbuffer;
1831 1942             $GETMSG(MSGID = .val_desc[dbg$li_value_value0],
1832 1943                 MSGLEN = msg_desc[dsc$b_length],
1833 1944                 BUFADR = msg_desc);
1834 1945             dbg$print(UPLIT BYTE(XASCIC '!AS'), msg_desc);
1835 1946             END;
1836 1947
1837 1948     [2,3]:
1838 1949         BEGIN
1839 1950             BIND format_tab = UPLIT BYTE(XASCIC '! ');
1840 1951             header_one = UPLIT BYTE(XASCIC 'CMP fP fPD IS CURMOD PRVMOD IPL');
1841 1952             header_two = UPLIT BYTE(XASCIC ' DV FU IV T N Z V C');
1842 1953             mode_names = UPLIT ('KRNL', 'EXEC', 'SUPR', 'USER') : VECTOR[4, LONG];
1843 1954
1844 1955             dbg$newline();
1845 1956             dbg$print(format_tab);
1846 1957             IF NOT .val_desc[dbg$vh_hdr_format] THEN dbg$print(header_one);
1847 1958             dbg$print(header_two);
1848 1959             dbg$newline();
1849 1960             dbg$print(format_tab);
1850 1961             IF NOT .val_desc[dbg$vh_hdr_format]
1851 1962             THEN dbg$print(UPLIT BYTE(XASCIC '!2UL!4UL!3UL!4UL !AD !AD!5UL'),
1852 1963                 .(val_desc[dbg$li_value_value0])<31,1,0>,
1853 1964                 .(val_desc[dbg$li_value_value0])<30,1,0>,
1854 1965                 .(val_desc[dbg$li_value_value0])<27,1,0>,
1855 1966                 .(val_desc[dbg$li_value_value0])<26,1,0>,
1856 1967                 4, mode_names[(val_desc[dbg$li_value_value0])<24,2,0>],
1857 1968                 4, mode_names[(val_desc[dbg$li_value_value0])<22,2,0>],
1858 1969                 .(val_desc[dbg$li_value_value0])<16,5,0>);
1859 1970
```

DBGVALUES
V04-000

K 15
16-Sep-1984 02:45:26 VAX-11 B111s-32 V4.0-742
14-Sep-1984 12:17:54 [DEBUG.SRC]DBGVALUES.B32;1

Page 68
(23)

: 1860	1971	4
: 1861	1972	4
: 1862	1973	4
: 1863	1974	4
: 1864	1975	4
: 1865	1976	4
: 1866	1977	4
: 1867	1978	4
: 1868	1979	4
: 1869	1980	4
: 1870	1981	4
: 1871	1982	4
: 1872	1983	4
: 1873	1984	4
: 1874	1985	4
: 1875	1986	4

```
dbg$print(UPLIT BYTE(%ASCIC '!3(3UL)!5(2UL)')
.(val_desc[dbg$_value_value0])<7,1,0>,
.(val_desc[dbg$_value_value0])<6,1,0>,
.(val_desc[dbg$_value_value0])<5,1,0>,
.(val_desc[dbg$_value_value0])<4,1,0>,
.(val_desc[dbg$_value_value0])<3,1,0>,
.(val_desc[dbg$_value_value0])<2,1,0>,
.(val_desc[dbg$_value_value0])<1,1,0>,
.(val_desc[dbg$_value_value0])<0,1,0>};

END;

[OTHERWISE]: SIGNAL(dbg$ interr,1,UPLIT BYTE(
%ASCIC 'DBGVALUES\DBG$PRINT_VALUE - unknown format code'));

TES;
RETURN;
END;
```

1877 1987 2
1878 1988 2
1879 1989 2
1880 1990 2
1881 1991 2
1882 1992 2
1883 1993 2
1884 1994 2
1885 1995 2
1886 1996 2
1887 1997 2
1888 1998 2
1889 1999 2
1890 2000 2
1891 2001 2
1892 2002 2
1893 2003 2
1894 2004 2
1895 2005 2
1896 2006 2
1897 2007 2
1898 2008 2
1899 2009 2
1900 2010 2
1901 2011 2
1902 2012 2
1903 2013 2
1904 2014 2
1905 2015 2
1906 2016 2
1907 2017 2
1908 2018 2
1909 2019 2
1910 2020 2
1911 2021 2
1912 2022 2
1913 2023 2
1914 2024 2
1915 2025 2
1916 2026 2
1917 2027 2
1918 2028 2
1919 2029 2
1920 2030 2
1921 2031 2
1922 2032 2
1923 2033 2
1924 2034 2
1925 2035 2
1926 2036 2
1927 2037 2
1928 2038 2
1929 2039 2
1930 2040 2
1931 2041 2
1932 2042 2
1933 2043 2

```
Radix will come in as something other than default if a radix was
explicitly specified in the command as in EX/HEX or if a radix
override was specified as in SET RADIX/OVERRIDE.
IF .radix NEQ dbg$sk_default
THEN
    dbg$print_value_as_integer(vms_desc,.radix)
ELSE
    BEGIN
        Unless this is a 'DEBUG' descriptor (created because a type override
        switch has been given), we first see if we can find any
        language-specific formatting rules.
        IF (.val_desc[dbg$b_dhdr_fcode] NEQ rst$sk_type_descr)
        OR (.val_desc[dbg$b_value_class] NEQ dsc$sk_class_2)
        THEN IF dbg$language_format(.val_desc) THEN RETURN;
        We get here if there are no language-specific formatting
        rules applicable to this data item, either because this
        is a 'DEBUG'-built value descriptor or because there are
        no applicable language-specific format exception entries.
        SELECTONE .val_desc[dbg$b_dhdr_fcode]
        OF SET
            [rst$sk_type_enum]:
                BEGIN
                    LOCAL
                        size,n_elems,elem_vect : REF VECTOR[.LONG];
                    dbg$sta_typ_enum(.val_desc[dbg$l_dhdr_typeid],n_elems,elem_vect,size);
                    INCR e FROM 0 TO .n_elems-1 DO
                        BEGIN
                            LOCAL adr_kind,adr_ptrs : VECTOR[3,.LONG];
                            dbg$sta_symvalue(.elem_vect[e],adr_ptrs,adr_kind);
                            IF .adr_kind NEQ dbg$sk_val_literal THEN SIGNAL(dbg$unimplent);
                            IF (.adr_ptrs[0]<.adr_ptrs[1],.size,0) EQL .val_desc[dbg$l_value_value0] THEN
                                BEGIN
                                    dbg$print_symbol_name(.elem_vect[e]);
                                    RETURN;
                                END;
                            END;
                        END;
                    Warn value out of range for enumeration type.
                    SIGNAL(dbg$enumrange);
                    dbg$print_value_as_integer(vms_desc);
                    END;
            [rst$sk_type_blifld]:
                BEGIN
                    LOCAL
                        count,
                        fields: REF VECTOR[.LONG];
                    fields = .val_desc[dbg$l_value_pointer];
```

```

1934 2044 4
1935 2045 4
1936 2046 4
1937 2047 3
1938 2048 3
1939 2049 3
1940 2050 4
1941 2051 4
1942 2052 4
1943 2053 4
1944 2054 3
1945 2055 3
1946 2056 3
1947 2057 3
1948 2058 3
1949 2059 3
1950 2060 3
1951 2061 3
1952 2062 3
1953 2063 3

```

```

count = .fields[0];
dbg$print(UPLIT BYTE(%ASCIC '['));
INCR e from 1 to .count-1 DO
    BEGIN
        dbg$print(UPLIT BYTE(%ASCIC '!UL'), .fields[e]);
        dbg$print(UPLIT BYTE(%ASCIC ','));
    END;

dbg$print(UPLIT BYTE(%ASCIC '!UL'), .fields[.count]);
dbg$print(UPLIT BYTE(%ASCIC ']'));
END;

[rst$sk_type_set]:
    dbg$print_set_value(.val_desc);

[rst$sk_type_file]:
    +f
    Just print the information that this is a file pointer.
    --
    dbg$print(UPLIT BYTE(%ASCIC 'file variable'));

```


1955	2064	3
1956	2065	4
1957	2066	4
1958	2067	4
1959	2068	4
1960	2069	4
1961	2070	4
1962	2071	4
1963	2072	4
1964	2073	4
1965	2074	4
1966	2075	4
1967	2076	4
1968	2077	4
1969	2078	4
1970	2079	4
1971	2080	4
1972	2081	4
1973	2082	5
1974	2083	5
1975	2084	5
1976	2085	5
1977	2086	5
1978	2087	5
1979	2088	4
1980	2089	4
1981	2090	4
1982	2091	5
1983	2092	5
1984	2093	5
1985	2094	5
1986	2095	5
1987	2096	5
1988	2097	4
1989	2098	4
1990	2099	4
1991	2100	5
1992	2101	5
1993	2102	5
1994	2103	5
1995	2104	5
1996	2105	5
1997	2106	5
1998	2107	5
1999	2108	4

[OTHERWISE]:

BEGIN

Here if there are no special formatting rules for this FCODE.
Just look at the DTYPE in the descriptor, and print the value
in a type-dependent format. However, we first have to ensure
that we have a valid CLASS field for LIB\$CVT_DX_DX.

IF .vms_desc[dsc\$b_class] EQL dsc\$k_class_2 THEN vms_desc[dsc\$b_class] = dsc\$k_class_s;
CASE .vms_desc[dsc\$b_dtype] FROM dbg\$k_minimum_dtype TO dbg\$k_maximum_dtype OF

SET

The first few case entries are for the various types of
ASCII text (ASCII, ASCII2, ASCIIW).
These all get translated to ASCII (DTYPE dsc\$k_dtype_t).

[dsc\$k_dtype_vt]:

BEGIN

vms_desc[dsc\$w_length] = (.vms_desc[dsc\$a_pointer])<0,16,0>;
vms_desc[dsc\$a_pointer] = 2+.vms_desc[dsc\$a_pointer];
vms_desc[dsc\$b_class] = dsc\$k_class_s;
vms_desc[dsc\$b_dtype] = dsc\$k_dtype_t;
dbg\$print_vms_value(vms_desc);
END;

[dsc\$k_dtype_ac]:

BEGIN

vms_desc[dsc\$w_length] = (.vms_desc[dsc\$a_pointer])<0,8,0>;
vms_desc[dsc\$a_pointer] = 1+.vms_desc[dsc\$a_pointer];
vms_desc[dsc\$b_class] = dsc\$k_class_s;
vms_desc[dsc\$b_dtype] = dsc\$k_dtype_t;
dbg\$print_vms_value(vms_desc);
END;

[dsc\$k_dtype_az]:

BEGIN

BUILTIN LOCC;
LOCAL length;
LOCC(%REF(0), vms_desc[dsc\$w_length], .vms_desc[dsc\$a_pointer]; length);
vms_desc[dsc\$w_length] = .vms_desc[dsc\$w_length] - .length;
vms_desc[dsc\$b_class] = dsc\$k_class_s;
vms_desc[dsc\$b_dtype] = dsc\$k_dtype_t;
dbg\$print_vms_value(vms_desc);
END;

```

2001 2109 4 [dsc$k_dtype_b,dsc$k_dtype_bu,dsc$k_dtype_w,dsc$k_dtype_wu,
2002 2110 4 dsc$k_dtype_l,dsc$k_dtype_lu,dsc$k_dtype_q,dsc$k_dtype_qu,
2003 2111 4 dsc$k_dtype_o,dsc$k_dtype_ou];
2004 2112 4 IF .dbg$gb_radix[dbg$b_radix_output] NEQ dbg$k_decimal
2005 2113 4 THEN
2006 2114 4     dbg$print_value_as_integer(vms_desc)
2007 2115 4 ELSE
2008 2116 4     dbg$print_vms_value(vms_desc, .sign_flag);
2009 2117 4
2010 2118 4 [dsc$k_dtype_f,dsc$k_dtype_d,dsc$k_dtype_g,dsc$k_dtype_h,
2011 2119 4 dsc$k_dtype_fc,dsc$k_dtype_dc,dsc$k_dtype_gc,dsc$k_dtype_hc,
2012 2120 4 dsc$k_dtype_nl,dsc$k_dtype_nlo,dsc$k_dtype_nr,dsc$k_dtype_nro,
2013 2121 4 dsc$k_dtype_nu,dsc$k_dtype_nz,dsc$k_dtype_p,dsc$k_dtype_f];
2014 2122 4     dbg$print_vms_value(vms_desc, .sign_flag);
2015 2123 4
2016 2124 4 [dsc$k_dtype_zi]:
2017 2125 4     dbg$ins_decode(.vms_desc[dsc$a_pointer],true,false);
2018 2126 4
2019 2127 4 [dsc$k_dtype_zem]:
2020 2128 4     dbg$ins_decode(.vms_desc[dsc$a_pointer],true,true);
2021 2129 4
2022 2130 4 [dsc$k_dtype_tf]:
2023 2131 6     dbg$print((Format_AC,(IF (.vms_desc[dsc$a_pointer])
2024 2132 5         THEN UPLIT BYTE(%ASCII 'True')
2025 2133 4         ELSE UPLIT BYTE(%ASCII 'False'))));
2026 2134 4
2027 2135 4 [dsc$k_dtype_adt]:
2028 2136 4     dbg$print_vms_value(vms_desc);
2029 2137 4
2030 2138 4 [dsc$k_dtype_dsc]:
2031 2139 4
2032 2140 4 [INRANGE,OUTRANGE]:
2033 2141 4     dbg$print_value_as_integer(vms_desc);
2034 2142 4 TES;
2035 2143 3     ! CASE .vms_desc[dsc$b_dtype]
2036 2144 3     ! SELECTONE .val_desc[dbg$b_dhdr_fcode]
2037 2145 2     ! End of 'dbg$print_value'
2038 2146 1

```

```

.PSECT DBG$PLIT,NOWRT, SHR, PIC,0
53 41 21 03 00081 P.AAL: .ASCII <3>\!AS\
5F 21 02 00085 P.AAM: .ASCII <2>\! \
49 20 44 4F 4D 56 52 50 20 44 4F 4D 52 55 43 00088 P.AAN: .ASCII <31>\CMP TP FPD IS CURMOD PRVMOD IPL\
4C 50 00097
20 4E 20 54 20 56 49 20 55 46 20 56 44 20 13 000A6
20 56 44 20 5A 000A8 P.AAO: .ASCII <19>\ DV FU IV T N Z V C\
4C 4E 52 4B 000B7
43 45 58 45 000BC P.AAP: .ASCII \KRNL\
52 50 55 53 000C0 .ASCII \EXEC\
52 45 53 55 000C4 .ASCII \SUPR\
52 45 53 55 000C8 .ASCII \USER\
34 21 4C 55 33 21 4C 55 34 21 4C 55 32 21 1F 000CC P.AAQ: .ASCII <31>\!2UL!4UL!3UL!4UL !AD !AD!5UL\
35 21 44 41 21 20 20 20 44 41 21 20 20 4C 55 000DB
4C 55 000EA

```

PC	Instruction	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418
----	-------------	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

			10	AE	9F	00088	PUSHAB	MSG_DESC	1945	
			FC	A8	9F	0008B	PUSHAB	P.AXL		
				026D	31	0008E	BRW	40\$		
		02		52	D1	00091	5\$:	CMPL	R2, #2	1948
				03	18	00094	BGEQ	7\$		
				0095	31	00096	6\$:	BRW	10\$	
		03		52	D1	00099	7\$:	CMPL	R2, #3	
				F8	14	0009C	BGTR	6\$		
		68		00	FB	0009E	CALLS	#0, DBG\$NEWLINE	1956	
				58	DD	000A1	PUSHL	R8	1957	
		69		01	FB	000A3	CALLS	#1, DBG\$PRINT		
		06		52	EB	000A6	BLBS	R2, 8\$	1958	
			03	A8	9F	000A9	PUSHAB	HEADER ONE		
		69		01	FB	000AC	CALLS	#1, DBG\$PRINT		
			23	A8	9F	000AF	8\$:	PUSHAB	HEADER TWO	1959
		69		01	FB	000B2	CALLS	#1, DBG\$PRINT		
		68		00	FB	000B5	CALLS	#0, DBG\$NEWLINE	1960	
				58	DD	000B8	PUSHL	R8	1961	
		69		01	FB	000BA	CALLS	#1, DBG\$PRINT		
		38		52	EB	000BD	BLBS	R2, 9\$	1962	
		51	20	A6	9E	000C0	MOVAB	32(R6), R1	1970	
7E	02	A1		00	EF	000C4	EXTZV	#0, #5, 2(R1), -(SP)		
50		61		16	EF	000CA	EXTZV	#22, #2, (R1), R0	1969	
			37	A840	DF	000CF	PUSHAL	MODE_NAMES[R0]		
				04	DD	000D3	PUSHL	#4		
50	03	A1	02	00	EF	000D5	EXTZV	#0, #2, 3(R1), R0	1968	
			37	A840	DF	000DB	PUSHAL	MODE_NAMES[R0]		
				04	DD	000DF	PUSHL	#4	1969	
7E		61	01	1A	EF	000E1	EXTZV	#26, #1, (R1), -(SP)		
7E		61	01	1B	EF	000E6	EXTZV	#27, #1, (R1), -(SP)		
7E		61	01	1E	EF	000EB	EXTZV	#30, #1, (R1), -(SP)		
7E		61	01	1F	EF	000F0	EXTZV	#31, #1, (R1), -(SP)		
			47	A8	9F	000F5	PUSHAB	P.AAQ	1963	
			69	0A	FB	000F8	CALLS	#10, DBG\$PRINT	1969	
			50	20	A6	9E	9\$:	MOVAB	32(R6), R0	1979
7E		60	01	00	EF	000FF	EXTZV	#0, #1, (R0), -(SP)		
7E		60	01	01	EF	00104	EXTZV	#1, #1, (R0), -(SP)	1978	
7E		60	01	02	EF	00109	EXTZV	#2, #1, (R0), -(SP)	1977	
7E		60	01	03	EF	0010E	EXTZV	#3, #1, (R0), -(SP)	1976	
7E		60	01	04	EF	00113	EXTZV	#4, #1, (R0), -(SP)	1975	
7E		60	01	05	EF	00118	EXTZV	#5, #1, (R0), -(SP)	1974	
7E		60	01	06	EF	0011D	EXTZV	#6, #1, (R0), -(SP)	1973	
7E		60	01	07	EF	00122	EXTZV	#7, #1, (R0), -(SP)	1972	
			67	A8	9F	00127	PUSHAB	P.AAR	1971	
			69	09	FB	0012A	CALLS	#9, DBG\$PRINT		
				04	0012D	RET			1931	
			76	A8	9F	0012E	10\$:	PUSHAB	P.AAS	1982
				01	DD	00131	PUSHL	#1		
				8F	DD	00133	PUSHL	#164706		
6A	00028362		03	FB	00139	CALLS	#3, LIB\$SIGNAL			
				04	0013C	RET			1930	
			01	08	AC	D1	11\$:	CMPL	RADIX, #1	1992
				0C	13	00141	BEQL	12\$		
			08	AC	DD	00143	PUSHL	RADIX	1994	
			F4	AD	9F	00146	PUSHAB	VMS_DESC		
0000V	CF			02	FB	00149	CALLS	#2, DBG\$PRINT_VALUE_AS_INTEGER		
				04	0014E	RET				

		03	06	A6	91	0014F	12\$:	CMPB	6(R6), #3	2002	
			05	12	00153			BNEQ	13\$		
			17	A6	95	00155		TSTB	23(R6)	2003	
				0D	13	00158		BEQL	14\$		
				56	DD	0015A	13\$:	PUSHL	R6	2004	
		00	01	FB	0015C			CALLS	#1, DBG\$LANGUAGE_FORMAT		
		01	50	E9	00163			BLBC	R0, 14\$		
				04	00166			RET			
		50	06	A6	9A	00167	14\$:	MOVZBL	6(R6), R0	2011	
		04		50	91	0016B		CMPB	R0, #4	2013	
				61	12	0016E		BNEQ	18\$		
				5E	DD	00170		PUSHL	SP	2018	
			08	AE	9F	00172		PUSHAB	ELEM_VECT		
			10	AE	9F	00175		PUSHAB	N_ELEMS		
			08	A6	DD	00178		PUSHL	8(R6)		
		00	04	FB	0017B			CALLS	#4, DBG\$STA_TYP_ENUM		
		52	01	CE	00182			MNEGL	#1, E	2024	
			39	11	00185			BRB	17\$		
			0C	AE	9F	00187	15\$:	PUSHAB	ADR_KIND	2022	
			E8	AD	9F	0018A		PUSHAB	ADR_PTRS		
			0C	BE42	DD	0018D		PUSHL	@ELEM_VECT[E]		
		00	03	FB	00191			CALLS	#3, DBG\$STA_SYMVALUE		
		01	0C	AE	D1	00198		CMPL	ADR_KIND, #T	2023	
			09	13	0019C			BEQL	16\$		
			8F	DD	0019E			PUSHL	#165888		
		6A	01	FB	001A4			CALLS	#1, LIB\$SIGNAL		
50	E8	BD	6E	EC	AD	EF	001A7	16\$:	EXTZV	ADR_PTRS+4, SIZE, @ADR_PTRS, R0	2024
		20	A6	50	D1	001AE		CMPL	R0, -32(R6)		
				0C	12	001B2		BNEQ	17\$		
			04	BE42	DD	001B4		PUSHL	@ELEM_VECT[E]	2026	
		00	01	FB	001B8			CALLS	#1, DBG\$PRINT_SYMBOL_NAME		
				04	001BF			RET		2025	
		52	08	AE	F2	001C0	17\$:	AOBLSS	N_ELEMS, E, 15\$	2019	
			8F	DD	001C5			PUSHL	#T65579	2033	
		6A	01	FB	001CB			CALLS	#1, LIB\$SIGNAL		
			00F0	31	001CE			BRW	32\$	2034	
		0E	50	91	001D1	18\$:		CMPB	R0, #14	2037	
			37	12	001D4			BNEQ	21\$		
		54	18	A6	D0	001D6		MOVL	24(R6), FIELDS	2043	
		53		64	D0	001DA		MOVL	(FIELDS), COUNT	2044	
			00A6	C8	9F	001DD		PUSHAB	P.AAT	2045	
		69	01	FB	001E1			CALLS	#1, DBG\$PRINT		
			52	D4	001E4			CLRL	E	2046	
			11	11	001E6			BRB	20\$		
			6442	DD	001E8	19\$:		PUSHL	(FIELDS)[E]	2048	
			00A8	C8	9F	001EB		PUSHAB	P.AAU		
		69	02	FB	001EF			CALLS	#2, DBG\$PRINT		
			00AC	C8	9F	001F2		PUSHAB	P.AAV	2049	
		69	01	FB	001F6			CALLS	#1, DBG\$PRINT		
		52	53	F2	001F9	20\$:		AOBLSS	COUNT, E, 19\$	2046	
			6443	DD	001FD			PUSHL	(FIELDS)[COUNT]	2052	
			00AF	C8	9F	00200		PUSHAB	P.AAW		
		69	02	FB	00204			CALLS	#2, DBG\$PRINT		
			00B3	C8	9F	00207		PUSHAB	P.AAX	2053	
				18	11	0020B		BRB	23\$		
		08	50	91	0020D	21\$:		CMPB	R0, #8	2056	
			0A	12	00210			BNEQ	22\$		

```
F 16
16-Sep-1984 02:45:26 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:17:54 [DEBUG.SRC]DBGVALUES.B32:1
```

: 2057

2059

2063

2072

2073

[illegible]

30 11 0028F

BRB

2141

F4	AD	F8	BD	B0	00291	278:	MOVW	@VMS_DESC+4, VMS_DESC	2083	
F8	AD		02	C0	00296		ADDL2	#2, VMS_DESC+4	2084	
			14	11	0029A		BRB	308	2085	
F4	AD	F8	BD	9B	0029C	288:	MOVZBW	@VMS_DESC+4, VMS_DESC	2092	
		F8	AD	D6	002A1		INCL	VMS_DESC+4	2093	
			0A	11	002A4		BRB	308	2094	
F8	BD									
F4	AD			00	3A	002A6	298:	LOCC	#0, VMS_DESC, @VMS_DESC+4	2103
F4	AD			50	A2	002AC		SUBW2	LENGTH, VMS_DESC	2104
F6	AD	010E		8F	B0	002B0	308:	MOVW	#270, VMS_DESC+2	2106
				4A	11	002B6		BRB	418	2107
		0A	00000000G	00	91	002B8	318:	CMPB	DBG\$GB_RADIX+1, #10	2112
				09	13	002BF		BEQL	338	
		F4		AD	9F	002C1	328:	PUSHAB	VMS_DESC	2114
0000V	CF			01	FB	002C4		CALLS	#1, -DBG\$PRINT_VALUE_AS_INTEGER	
					04	002C9		RET		
		F4		57	DD	002CA	338:	PUSHL	SIGN FLAG	2122
0000V	CF			AD	9F	002CC		PUSHAB	VMS_DESC	
				02	FB	002CF		CALLS	#2, -DBG\$PRINT_VMS_VALUE	
					04	002D4		RET		
				7E	D4	002D5	348:	CLRL	-(SP)	2125
				02	11	002D7		BRB	368	
				01	DD	002D9	358:	PUSHL	#1	2128
				01	DD	002DB	368:	PUSHL	#1	
		F8		AD	DD	002DD		PUSHL	VMS_DESC+4	
00000000G	00			03	FB	002E0		CALLS	#3, -DBG\$INS_DECODE	
					04	002E7		RET		
	07	F8	BD	E9	002E8	378:	BLBC	@VMS_DESC+4, 388	2131	
	50	00C3	C8	9E	002EC		MOVAB	P.AAZ, R0	2132	
			05	11	002F1		BRB	398		
	50	00C8	C8	9E	002F3	388:	MOVAB	P.ABA, R0	2133	
			50	DD	002F8	398:	PUSHL	R0		
		FF7B	C8	9F	002FA		PUSHAB	FORMAT AC	2131	
	69		02	FB	002FE	408:	CALLS	#2, -DBG\$PRINT		
					04	00301		RET		
		F4	AD	9F	00302	418:	PUSHAB	VMS_DESC	2136	
0000V	CF		01	FB	00305		CALLS	#1, -DBG\$PRINT_VMS_VALUE		
					04	0030A		RET	2146	

; Routine Size: 779 bytes, Routine Base: DBG\$CODE + 1116

```
2040 2147 1 GLOBAL ROUTINE DBG$PRINT_VALUE_AS_INTEGER(vms_desc: REF dbg$stg_desc) : NOVALUE =
2041 2148 BEGIN
2042 2149 BUILTIN ACTUALCOUNT,ACTUALPARAMETER,MOVCS;
2043 2150 LOCAL
2044 2151 radix,
2045 2152 radix_override_flag, ! TRUE if radix override was applied
2046 2153 data_bytes,
2047 2154 byte_size,
2048 2155 data_size,
2049 2156 data_addr,
2050 2157 data_buff : VECTOR [512*4,BYTE],
2051 2158 digit_count,
2052 2159 digit_value : BYTE,
2053 2160 text_index,
2054 2161 text_buff : VECTOR [512*9,BYTE];
2055 2162
2056 2163 BIND digit = UPLIT BYTE('0123456789ABCDEF') : VECTOR [16,BYTE];
2057 2164
2058 2165 radix_override_flag = FALSE;
2059 2166 IF actualcount() GTR 1
2060 2167 THEN
2061 2168 BEGIN
2062 2169 radix = actualparameter(2);
2063 2170 IF .radix EQL dbg$k_default
2064 2171 THEN
2065 2172 radix = dbg$nget_radix()
2066 2173 ELSE
2067 2174 radix_override_flag = TRUE;
2068 2175 END
2069 2176 ELSE
2070 2177 radix = dbg$nget_radix();
2071 2178
2072 2179 text_index = 512*9;
2073 2180
2074 2181 data_addr = .vms_desc[dsc$a_pointer];
2075 2182 IF (data_size = dbg$data_length(.vms_desc)) GTR 512*%BPUNIT THEN
2076 2183 BEGIN
2077 2184 ! *** SIGNAL(truncation) *****
2078 2185 data_size = 512*%BPUNIT;
2079 2186 END;
2080 2187 data_bytes = (.data_size + (%BPUNIT-1))/%BPUNIT;
2081 2188 MOVCS(data_bytes,.data_addr,%REF(0),%REF(512*4),data_buff);
2082 2189 IF (.data_size AND (%BPUNIT-1)) NEQ 0
2083 2190 THEN data_buff[.data_bytes-1] =
2084 2191 .data_buff[.data_bytes-1] AND NOT (-1*(.data_size AND (%BPUNIT-1)));
2085 2192
2086 2193 SELECTONE .radix OF
2087 2194 SET
2088 2195 [dbg$k_decimal]:
2089 2196 BEGIN
2090 2197 SELECTONE .vms_desc[dsc$b_dtype] OF
2091 2198 SET
2092 2199 [dsc$k_dtype_bu,dsc$k_dtype_wu,
2093 2200 dsc$k_dtype_lu,dsc$k_dtype_qu,
2094 2201 dsc$k_dtype_ou,dsc$k_dtype_z,
2095 2202 dsc$k_dtype_v,dsc$k_dtype_vu]:
2096 2203 IF .radix_override_flag
```



```
2097 2204 THEN
2098 2205 IF (.data_buff)<.data_size-1,1,0> THEN
2099 2206 BEGIN
2100 2207 dbg$print(Format_AD,1,UPLIT BYTE('-'));
2101 2208 INCR m FROM 0 TO .data_bytes-1 DO
2102 2209 IF .data_buff[m] NEQ 0 THEN
2103 2210 BEGIN
2104 2211 data_buff[m] = -.data_buff[m];
2105 2212 INCR n FROM .m+1 TO .data_bytes-1 DO
2106 2213 data_buff[n] = NOT .data_buff[n];
2107 2214 EXITLOOP;
2108 2215 END;
2109 2216 IF (.data_size AND (XBPUNIT-1)) NEQ 0
2110 2217 THEN data_buff[.data_bytes-1] =
2111 2218 .data_buff[.data_bytes-1] AND NOT (-1^(.data_size AND (XBPUNIT-1)));
2112 2219 END;
2113 2220 [OTHERWISE]:
2114 2221 IF (.data_buff)<.data_size-1,1,0> THEN
2115 2222 BEGIN
2116 2223 dbg$print(Format_AD,1,UPLIT BYTE('-'));
2117 2224 INCR m FROM 0 TO .data_bytes-1 DO
2118 2225 IF .data_buff[m] NEQ 0 THEN
2119 2226 BEGIN
2120 2227 data_buff[m] = -.data_buff[m];
2121 2228 INCR n FROM .m+1 TO .data_bytes-1 DO
2122 2229 data_buff[n] = NOT .data_buff[n];
2123 2230 EXITLOOP;
2124 2231 END;
2125 2232 IF (.data_size AND (XBPUNIT-1)) NEQ 0
2126 2233 THEN data_buff[.data_bytes-1] =
2127 2234 .data_buff[.data_bytes-1] AND NOT (-1^(.data_size AND (XBPUNIT-1)));
2128 2235 END;
2129 2236 TES;
2130 2237 WHILE (data_size = .data_bytes) GTR 0 DO
2131 2238 BEGIN
2132 2239 LOCAL digit_val;
2133 2240 data_bytes = 0;
2134 2241 digit_val = 0;
2135 2242 DECR d FROM .data_size-1 TO 0 DO
2136 2243 BEGIN
2137 2244 digit_val = (.digit_val*8)+.data_buff[d];
2138 2245 IF (data_buff[d] = -.digit_val/10) NEQ 0
2139 2246 THEN IF .data_bytes EQL 0 THEN data_bytes = .d+1;
2140 2247 digit_val = .digit_val - 10*(.digit_val/10);
2141 2248 END;
2142 2249 text_buff[(text_index=.text_index-1)] = .digit_val<0,8,0>+'0';
2143 2250 END;
2144 2251 dbg$print(Format_AD,512*9-.text_index,text_buff[.text_index]);
2145 2252 RETURN;
2146 2253 END;
2147 2254 [dbg$k_binary]: byte_size = 1;
2148 2255 [dbg$k_octal]: byte_size = 3;
2149 2256 [dbg$k_hex]: byte_size = 4;
2150 2257 [OTHERWISE]:
2151 2258
2152 2259
2153 2260
```

```

: 2154      2261      2      TES;
: 2155      2262      2
: 2156      2263      2      digit_count = (.data_size + (.byte_size-1))/ .byte_size;
: 2157      2264      2
: 2158      2265      2      INCR index FROM 0 TO .digit_count-1 DO
: 2159      2266      2      BEGIN
: 2160      2267      2      IF (.byte_size NEQ 3) AND ((.index AND 7) EQL 0) AND (.index NEQ 0)
: 2161      2268      2      THEN text_buff[(text_index = .text_index-1)] = ' ';
: 2162      2269      2      digit_value = .digit[(data_buff)<.index*.byte_size,.byte_size,0>];
: 2163      2270      2      text_buff[(text_index = .text_index-1)] = .digit_value;
: 2164      2271      2      END;
: 2165      2272      2
: 2166      2273      2      IF .digit_value GTRU '9' THEN text_buff[(text_index = .text_index-1)] = '0';
: 2167      2274      2
: 2168      2275      2      dbg$print(format_AD,512*9-.text_index,text_buff[.text_index]);
: 2169      2276      2      END;
:                               ! End of routine 'dbg$print_value_as_integer'
```

```

                                .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0
45 44 43 42 41 39 38 37 36 35 34 33 32 31 30 00153 P.ABB: .ASCII  \0123456789ABCDEF\
                                46 00162
                                2D 00163 P.ABC: .ASCII  \-\
                                2D 00164 P.ABD: .ASCII  \-\
                                DIGIT=          P.ABB

                                .PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0
                                OFFC 00000
                                .ENTRY  DBG$PRINT VALUE AS INTEGER, Save R2,R3,R4,-
                                R5,R6,R7,R8,R9,R10,R11
                                MOVAB   -5124(SP), SP
                                CLRL    RADIX_OVERRIDE_FLAG
                                CMPB    (AP), #1
                                BLEQU   1$
                                MOVL    8(AP), RADIX
                                CMPL    RADIX, #1
                                BEQL    1$
                                MOVL    #1, RADIX_OVERRIDE_FLAG
                                BRB     2$
                                CALLS   #0, DBG$NGET_RADIX
                                MOVL    R0, RADIX
                                MOVZWL  #4096, TEXT_INDEX
                                MOVL    VMS_DESC, R8
                                MOVL    4(R8), DATA_ADDR
                                PUSHL   R8
                                CALLS   #1, DBG$DATA_LENGTH
                                MOVL    R0, DATA_SIZE
                                CMPL    DATA_SIZE, #4096
                                BLEQ    3$
                                MOVZWL  #4096, DATA_SIZE
                                MOVAB   7(R9), R0
                                DIVL3   #8, R0, DATA_BYTES
                                MOVC5   DATA_BYTES, (DATA_ADDR), #0, #516, -
                                DATA_BUFF
                                2147
                                2165
                                2166
                                2169
                                2170
                                2174
                                2166
                                2177
                                2179
                                2181
                                2182
                                2185
                                2187
                                2188
```

				52	D4	0005D	CLRL	R2		2189
		07		59	93	0005F	BITB	DATA_SIZE, #7		
				15	13	00062	BEQL	4\$		
				52	D6	00064	INCL	R2		
50	59	03		00	EF	00066	EXTZV	#0, #3, DATA_SIZE, R0	2191	
	50	8F		50	78	00068	ASHL	R0, #-1, R0		
		FDFB	CD46	50	8A	00073	BICB2	R0, DATA_BUFF-1[DATA_BYTES]		
		0A		5A	D1	00079	4\$: CMPL	RADIX, #T0	2195	
				03	13	0007C	BEQL	5\$		
				0109	31	0007E	BRW	24\$		
				50	A8	9A	5\$: MOVZBL	2(R8), R0	2197	
				05	50	91	CMPB	R0, #5	2199	
					0A	1B	BLEQU	6\$		
					50	91	CMPB	R0, #25		
				19	05	13	BEQL	6\$		
					50	91	CMPB	R0, #34		
				22	52	12	BNEQ	13\$		
				03	5B	E8	6\$: BLBS	RADIX_OVERRIDE_FLAG, 8\$	2203	
					00AC	31	7\$: BRW	19\$		
					A9	9E	8\$: MOVAB	-1(R9), R0	2205	
F3					50	E1	BBC	R0, DATA_BUFF, 7\$		
					FDFC	CD	PUSHAB	P.ABC	2207	
					00000000'	EF	PUSHL	#1		
						01	PUSHAB	FORMAT AD		
					00000000'	EF	CALLS	#3, DBG\$PRINT		
						03	MNEGL	#1, M	2208	
						01	BRB	12\$		
						22	MOVZBL	DATA_BUFF[M], R0	2209	
						50	BEQL	12\$		
						1A	MNEGB	R0, DATA_BUFF[M]	2211	
						50	MOVL	M, N	2212	
						51	BRB	11\$		
						D0	MCOMB	DATA_BUFF[N], DATA_BUFF[N]	2213	
						09	AOBLS	DATA_BYTES, N, 10\$		
						11	BRB	18\$	2210	
						F2	AOBLS	DATA_BYTES, M, 9\$	2209	
						00DE	BRB	18\$	2216	
						56	MOVAB	-1(R9), R0	2222	
						4A	BBC	R0, DATA_BUFF, 19\$		
						11	PUSHAB	P.ABD	2224	
						00E4	PUSHL	#1		
						A9	PUSHAB	FORMAT AD		
						9E	CALLS	#3, DBG\$PRINT		
						00E6	MNEGL	#1, M	2225	
						E1	BRB	17\$		
						00EA	MOVZBL	DATA_BUFF[M], R0	2226	
						EF	BEQL	17\$		
						01	MNEGB	R0, DATA_BUFF[M]	2228	
						DD	MOVL	M, N	2229	
						00F0	BRB	16\$		
						00F6	MCOMB	DATA_BUFF[N], DATA_BUFF[N]	2230	
						00F8	AOBLS	DATA_BYTES, N, 15\$		
						00FE	BRB	18\$	2227	
						0105	AOBLS	DATA_BYTES, M, 14\$	2226	
						0108	BLBC	R2, T9\$	2233	
						9A	EXTZV	#0, #3, DATA_SIZE, R0	2235	
						1A	ASHL	R0, #-1, R0		
						13				
						50				
						8E				
						D0				
						11				
						92				
						F2				
						11				
						F2				
						00130				
						00133				
						00138				

	FDFB CD46	50	8A 00140		BICB2	R0, DATA_BUFF-1[DATA_BYTES]	
	59	56	D0 00146	198:	MOVL	DATA_BYTES, DATA_SIZE	2239
		03	14 00149		BGTR	208	
		009C	31 00148		BRW	318	
		56	D4 0014E	208:	CLRL	DATA_BYTES	2242
		51	D4 00150		CLRL	DIGIT_VAL	2243
	50	59	D0 00152		MOVL	DATA_SIZE, D	2244
		29	11 00155		BRB	238	
52	51	08	78 00157	218:	ASHL	#8, DIGIT_VAL, R2	2246
	51	FDFC CD40	9A 0015B		MOVZBL	DATA_BUFF[D], DIGIT_VAL	
	51	52	C0 00161		ADDL2	R2, DIGIT_VAL	
52	51	0A	C7 00164		DIVL3	#10, DIGIT_VAL, R2	2247
	FDFC CD40	52	90 00168		MCVB	R2, DATA_BUFF[D]	
		52	D5 0016E		TSTL	R2	
		08	13 00170		BEQL	228	
		56	D5 00172		TSTL	DATA_BYTES	2248
		04	12 00174		BNEQ	228	
	56	01	A0 9E 00176		MOVAB	1(R0), DATA_BYTES	
	52	0A	C4 0017A	228:	MULL2	#10, R2	2249
	51	52	C2 0017D		SUBL2	R2, DIGIT_VAL	
	D4	50	F4 00180	238:	SOBGEQ	D, 218	2244
774E	51	30	81 00183		ADDB3	#48, DIGIT_VAL, TEXT_BUFF[SP]	2251
		BC	11 00188		BRB	198	2239
	02	5A	D1 0018A	248:	CMPL	RADIX, #2	2257
		05	12 0018D		BNEQ	258	
	53	01	D0 0018F		MOVL	#1, BYTE_SIZE	
		12	11 00192		BRB	278	
	08	5A	D1 00194	258:	CMPL	RADIX, #8	2258
		05	12 00197		BNEQ	268	
	53	03	D0 00199		MOVL	#3, BYTE_SIZE	
		08	11 0019C		BRB	278	
	10	5A	D1 0019E	268:	CMPL	RADIX, #16	2259
		03	12 001A1		BNEQ	278	
	53	04	D0 001A3		MOVL	#4, BYTE_SIZE	
	50	FF A349	9E 001A6	278:	MOVAB	-1(BYTE_SIZE)[DATA_SIZE], R0	2263
55	50	53	C7 001AB		DIVL3	BYTE_SIZE, R0, DIGIT_COUNT	
	50	01	CE 001AF		MNEGL	#1, INDEX	2265
		29	11 001B2		BRB	308	
	03	53	D1 001B4	288:	CMPL	BYTE_SIZE, #3	2267
		0D	13 001B7		BEQL	298	
	07	50	93 001B9		BITB	INDEX, #7	
		08	12 001BC		BNEQ	298	
		50	D5 001BE		TSTL	INDEX	
		04	13 001C0		BEQL	298	
	774E	20	90 001C2		MOVB	#32, TEXT_BUFF[SP]	2268
	50	53	C5 001C6	298:	MULL3	BYTE_SIZE, INDEX, R2	2269
51	FDFC	52	EF 001CA		EXTZV	R2, BYTE_SIZE, DATA_BUFF, R1	
	CD	54	90 001D1		MOVB	DIGIT[R1], DIGIT_VALUE	
		774E	90 001D9		MOVB	DIGIT_VALUE, TEXT_BUFF[SP]	2270
		50	F2 001DD	308:	AOBLSS	DIGIT_COUNT, INDEX, 288	2265
	D3	54	91 001E1		CMPB	DIGIT_VALUE, #57	2273
		04	1B 001E4		BLEQU	318	
	774E	30	90 001E6		MOVB	#48, TEXT_BUFF[SP]	
		6E47	9F 001EA	318:	PUSHAB	TEXT_BUFF[TEXT_INDEX]	2275
		EE00	9F 001ED		PUSHAB	-4608(TEXT_INDEX)	
	6E	6E	CE 001F1		MNEGL	(SP), (SP)	
		00000000	EF 9F 001F4		PUSHAB	FORMAT_AD	

DBGVALUES
V04-000

M 16
16-Sep-1984 02:45:26 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:17:54 [DEBUG.SRC]DBGVALUES.B32;1

Page 83
(27)

00000000G 00

03 FB 001FA
04 00201

CALLS #3, DBG\$PRINT
RET

: 2276

; Routine Size: 514 bytes, Routine Base: DBG\$CODE + 1421

```

2171 2277 1 ROUTINE DBG$PRINT_VMS_VALUE(vms_desc: REF dbg$stg_desc) : NOVALUE =
2172 2278 BEGIN
2173 2279 BUILTIN ACTUALCOUNT,ACTUALPARAMETER;
2174 2280 BIND exp_zero = UPLIT BYTE('E+0000'); ! Hack to fix FOR$CVT bug
2175 2281 LOCAL
2176 2282     local_desc      : dbg$stg_desc,
2177 2283     buffer_desc     : dbg$stg_desc,
2178 2284     status,
2179 2285     text_buffer     : VECTOR [64,BYTE],
2180 2286     text_length     : WORD;
2181 2287
2182 2288 ch$move(12, vms_desc, local_desc);
2183 2289 buffer_desc[dsc$b_class] = dsc$k_class_s;
2184 2290 buffer_desc[dsc$b_dtype] = dsc$k_dtype_t;
2185 2291 buffer_desc[dsc$b_length] = 64;
2186 2292 buffer_desc[dsc$a_pointer] = text_buffer;
2187 2293
2188 2294 SELECTONE .vms_desc[dsc$b_dtype] OF
2189 2295 SET
2190 2296 [dsc$k_dtype_fc, dsc$k_dtype_dc, dsc$k_dtype_gc, dsc$k_dtype_hc]:
2191 2297 BEGIN
2192 2298     local_desc[dsc$b_length] = .local_desc[dsc$b_length]/2;
2193 2299     local_desc[dsc$b_dtype] = .local_desc[dsc$b_dtype] -2;
2194 2300     local_desc[dsc$b_class] = dsc$k_class_s;
2195 2301     dbg$print(FORMAT_AD,1,UPLIT BYTE('('));
2196 2302     dbg$print vms value(local_desc);
2197 2303     dbg$print(FORMAT_AD,1,UPLIT BYTE(','));
2198 2304     local_desc[dsc$a_pointer] = .local_desc[dsc$a_pointer] + .local_desc[dsc$b_length];
2199 2305     dbg$print vms value(local_desc);
2200 2306     dbg$print(FORMAT_AD,1,UPLIT BYTE(')'));
2201 2307 END;
2202 2308
2203 2309 [dsc$k_dtype_f]:
2204 2310 BEGIN
2205 2311 BUILTIN CVTFD;
2206 2312 LOCAL dvalue : BLOCK[8,BYTE];
2207 2313 LOCAL digits,spaces,length;
2208 2314 BUILTIN SKPC,LOCC;
2209 2315
2210 2316 ! Use the FORTRAN 'G' format routine, which only prints the
2211 2317 ! answer in exponential form if it has to.
2212 2318 ! Since there is not FOR$CVT_F_TG routine, convert the source
2213 2319 ! to d_float.
2214 2320
2215 2321 CVTFD(.local_desc[dsc$a_pointer], dvalue);
2216 2322 IF NOT for$cvf_d_tg( dvalue,
2217 2323     buffer_desc,
2218 2324     7, ! Significant digits
2219 2325     0, ! Scale factor
2220 2326     1, ! Digits before decimal point
2221 2327     ! in exponential form
2222 2328     2) ! Digits after 'E'
2223 2329     ! in exponential form
2224 2330 THEN
2225 2331     SDBG_ERROR('DBGVALUES\DBG$PRINT_VMS_VALUE');
2226 2332
2227 2333 ! The result is right-justified. Find the position where

```

```
2228      ! the result begins (the first non-blank character in the
2229      ! buffer) and print the result.
2230      SKPC(%REF(' '),%REF(64),text_buffer,length,digits);
2231      LOCC(%REF(' '),length,.digits,.spaces);
2232      IF ch$eq(4,.spaces-4,4,exp_zero) THEN spaces = .spaces-4;
2233      IF (actualcount() GTR 1 AND actualparameter(2))
2234      AND (.(.digits)<0,8,0> NEQ '-')
2235      THEN DBG$PRINT(format_AD,1,UPLIT BYTE('+'));
2236      dbg$print(format_AD, .spaces - .digits, .digits);
2237      END;
2238
2239      [dsc$k_dtype_d]:
2240      BEGIN
2241      LOCAL digits,spaces,length;
2242      BUILTIN SKPC,LOCC;
2243
2244      ! Use the FORTRAN "G" format routine, which only prints the
2245      ! answer in exponential form if it has to.
2246      IF NOT for$cvt_d_tg(.local_desc[dsc$a_pointer],
2247      buffer_desc,
2248      16,          ! Significant digits
2249      0,           ! Scale factor
2250      1,          ! Digits before decimal point
2251      2)          ! Digits after "E"
2252      THEN
2253      $DBG_ERROR('DBGVALUES\DBG$PRINT_VMS_VALUE');
2254
2255      ! The result is right-justified. Find the position where
2256      ! the result begins (the first non-blank character in the
2257      ! buffer) and print the result.
2258      SKPC(%REF(' '),%REF(64),text_buffer,length,digits);
2259      LOCC(%REF(' '),length,.digits,.spaces);
2260      IF ch$eq(4,.spaces-4,4,exp_zero) THEN spaces = .spaces-4;
2261      IF (actualcount() GTR 1 AND actualparameter(2))
2262      AND (.(.digits)<0,8,0> NEQ '-')
2263      THEN DBG$PRINT(format_AD,1,UPLIT BYTE('+'));
2264      dbg$print(format_AD, .spaces - .digits, .digits);
2265      END;
2266
2267      [dsc$k_dtype_g]:
2268      BEGIN
2269      LOCAL digits,spaces,length;
2270      BUILTIN SKPC,LOCC;
2271
2272      ! Use the FORTRAN "G" format routine, which only prints the
2273      ! answer in exponential form if it has to.
2274      IF NOT for$cvt_g_tg(.local_desc[dsc$a_pointer],
2275      buffer_desc,
2276      15,          ! Significant digits
2277      0,           ! Scale factor
2278      1,          ! Digits before decimal point
```

```

2285      2391
2286      2392
2287      2393
2288      2394
2289      2395
2290      2396
2291      2397
2292      2398
2293      2399
2294      2400
2295      2401
2296      2402
2297      2403
2298      2404
2299      2405
2300      2406
2301      2407
2302      2408
2303      2409
2304      2410
2305      2411
2306      2412
2307      2413
2308      2414
2309      2415
2310      2416
2311      2417
2312      2418
2313      2419
2314      2420
2315      2421
2316      2422
2317      2423
2318      2424
2319      2425
2320      2426
2321      2427
2322      2428
2323      2429
2324      2430
2325      2431
2326      2432
2327      2433
2328      2434
2329      2435
2330      2436
2331      2437
2332      2438
2333      2439
2334      2440
2335      2441
2336      2442
2337      2443
2338      2444
2339      2445
2340      2446
2341      2447

      3)          ! in exponential form
                  ! Digits after 'E'
                  ! in exponential form
THEN
  $DBG_ERROR('DBGVALUES\DBG$PRINT_VMS_VALUE');

  ! The result is right-justified. Find the position where
  ! the result begins (the first non-blank character in the
  ! buffer) and print the result.
  SKPC(XREF(' '),XREF(64),text_buffer,length,digits);
  LOCC(XREF(' '),length,digits,spaces);
  IF ch$eq(5,.spaces-5,exp_zero) THEN spaces = .spaces-5;
  IF (actualcount() GTR 1 AND actualparameter(2))
    AND (.(digits)<0,8,0> NEQ '-')
    THEN DBG$PRINT(format_AD,1,UPLIT BYTE('+'));
  dbg$print(format_AD, .spaces - .digits, .digits);
END;

[dsc$k_dtype_h]:
BEGIN
  LOCAL digits,spaces,length;
  BUILTIN SKPC,LOCC;

  ! Use the FORTRAN 'G' format routine, which only prints the
  ! answer in exponential form if it has to.
  IF NOT for$cvr_h_tg(.local_desc[dsc$a_pointer],
    buffer_desc,
    33,          ! Significant digits
    0,           ! Scale factor
    1,           ! Digits before decimal point
                ! in exponential form
    4)          ! Digits after 'E'
                ! in exponential form
  THEN
    $DBG_ERROR('DBGVALUES\DBG$PRINT_VMS_VALUE');

    ! The result is right-justified. Find the position where
    ! the result begins (the first non-blank character in the
    ! buffer) and print the result.
    SKPC(XREF(' '),XREF(64),text_buffer,length,digits);
    LOCC(XREF(' '),length,digits,spaces);
    IF ch$eq(6,.spaces-6,exp_zero) THEN spaces = .spaces-6;
    IF (actualcount() GTR 1 AND actualparameter(2))
      AND (.(digits)<0,8,0> NEQ '-')
      THEN DBG$PRINT(format_AD,1,UPLIT BYTE('+'));
    dbg$print(format_AD, .spaces - .digits, .digits);
  END;

[dsc$k_dtype_t]:
BEGIN
  BUILTIN
  PROBER;
  LOCAL
  addr,

```


2342 2448
2343 2449
2344 2450
2345 2451
2346 2452
2347 2453
2348 2454
2349 2455
2350 2456
2351 2457
2352 2458
2353 2459
2354 2460
2355 2461
2356 2462
2357 2463
2358 2464
2359 2465
2360 2466
2361 2467
2362 2468
2363 2469
2364 2470
2365 2471
2366 2472
2367 2473
2368 2474
2369 2475
2370 2476
2371 2477
2372 2478
2373 2479
2374 2480
2375 2481
2376 2482
2377 2483
2378 2484
2379 2485
2380 2486
2381 2487
2382 2488
2383 2489
2384 2490
2385 2491
2386 2492
2387 2493
2388 2494
2389 2495
2390 2496
2391 2497
2392 2498
2393 2499
2394 2500
2395 2501
2396 2502
2397 2503
2398 2504

```

        bytes;
local_desc[dsc$w_length] = MIN(.local_desc[dsc$w_length],2048);

! Check for read access to the address. There are certain
! cases where we can get here with the address in the descriptor
! not readable. One example is EXAMINE/ASCII X, where X points
! to the descriptor
! 0000FFFF
! 00000000
! In this case the VMS descriptor came from a volatile value
! descriptor, and the address '0' in the descriptor was never
! checked for readability.
addr = .local_desc[dsc$a_pointer];
bytes = .local_desc[dsc$w_length];
IF NOT PROBER(%REF(0),bytes,.addr)
THEN
    SIGNAL(dbg$_noaccessr,1,.addr);

dbg$print(UPLOT BYTE(%ASCII "'!AF'"),.local_desc[dsc$w_length],
        .local_desc[dsc$a_pointer]);
END;

[OTHERWISE]:
BEGIN
!+
! Somewhat of a hack for FORTRAN - the FORTRAN compiler gives
! us types BU, WU, LU for LOGICAL variables even though they
! are really treated as signed integers. Change the dtype here
! so we print them right.
!-
IF .dbg$gb_language EQL dbg$_fortran
THEN
    IF .local_desc[dsc$b_dtype] EQL dsc$_k_dtype_bu
    THEN local_desc[dsc$b_dtype] = dsc$_k_dtype_b
    ELSE IF .local_desc[dsc$b_dtype] EQL dsc$_k_dtype_wu
    THEN local_desc[dsc$b_dtype] = dsc$_k_dtype_w
    ELSE IF .local_desc[dsc$b_dtype] EQL dsc$_k_dtype_lu
    THEN local_desc[dsc$b_dtype] = dsc$_k_dtype_l;

dbg$cvl_dx_dx(local_desc,buffer_desc,text_length);

IF .signed_dtype[.local_desc[dsc$b_dtype]] AND
(actualcount() GTR 1 AND actualparameter(2)) AND
(.text_buffer[0] NEQ '-') AND (.text_buffer[0] NEQ '+') THEN
    BEGIN
        ch$move(.text_length,text_buffer[0],text_buffer[1]);
        text_buffer[0] = '+';
        text_length = .text_length + 1;
    END;

dbg$print(format_AD,.text_length,text_buffer);
END;
TES;
END;
! End of dbg$print_vms_value

```

```
                                .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0
                                30  30  30  30  2B  45  00165 P.ABE:  .ASCII  \E+0000\
                                28  0016B P.ABF:  .ASCII  \(\
                                2C  0016C P.ABG:  .ASCII  \)\
                                29  0016D P.ABH:  .ASCII  \)\
24  47  42  44  5C  53  45  55  4C  41  56  47  42  44  1D  0016E P.ABI:  .ASCII  <29>\DBGVALUES\<92>\DBG$PRINT_VMS_VALU\
    55  4C  41  56  5F  53  4D  56  5F  54  4E  49  52  50  0017D
                                45  0018B      .ASCII  \E\
                                28  0018C P.ABJ:  .ASCII  \+\
24  47  42  44  5C  53  45  55  4C  41  56  47  42  44  1D  0018D P.ABK:  .ASCII  <29>\DBGVALUES\<92>\DBG$PRINT_VMS_VALU\
    55  4C  41  56  5F  53  4D  56  5F  54  4E  49  52  50  0019C
                                45  001AA      .ASCII  \E\
                                28  001AB P.ABL:  .ASCII  \+\
24  47  42  44  5C  53  45  55  4C  41  56  47  42  44  1D  001AC P.ABM:  .ASCII  <29>\DBGVALUES\<92>\DBG$PRINT_VMS_VALU\
    55  4C  41  56  5F  53  4D  56  5F  54  4E  49  52  50  001BB
                                45  001C9      .ASCII  \E\
                                28  001CA P.ABN:  .ASCII  \+\
24  47  42  44  5C  53  45  55  4C  41  56  47  42  44  1D  001CB P.ABO:  .ASCII  <29>\DBGVALUES\<92>\DBG$PRINT_VMS_VALU\
    55  4C  41  56  5F  53  4D  56  5F  54  4E  49  52  50  001DA
                                45  001E8      .ASCII  \E\
                                28  001E9 P.ABP:  .ASCII  \+\
                                22  46  41  21  22  05  001EA P.ABQ:  .ASCII  <5>\''!AF''\
                                EXP_ZERO=      P.ABE
```

```
                                .PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0
                                07FC 00000 DBG$PRINT_VMS_VALUE:
                                .WORD  Save R2,R3,R4,R5,R6,R7,R8,R9,R10
                                5A 00000000G 00 9E 00002 MOVAB  FOR$CVT D IG, R10
                                59 00000000G 00 9E 00009 MOVAB  LIB$SIGNAC, R9
                                58 00000000G 00 9E 00010 MOVAB  DBG$PRINT, R8
                                57 00000000G EF 9E 00017 MOVAB  FORMAT AD, R7
                                5E      9C AE 9E 0001E MOVAB  -100(SP), SP
                                56      04 AC D0 00022 MOVL   VMS_DESC, R6
                                5B AE 010E0040 0C 28 00026 MOVCL  #12, (R6), LOCAL_DESC
                                50 AE      0C AE 9E 0002B MOVL   #17694784, BUFFER_DESC
                                50      02 A6 9A 0003B MOVZBL 2(R6), R0
                                0C      05 50 91 0003C CMPB   R0, #12
                                0D      0A 50 91 00041 BLSSU  R0, #13
                                1D      4E 50 91 00046 1$: CMPB   R0, #29
                                1E      49 50 91 00049 BLSSU  R0, #30
                                51      58 AE 3C 00050 2$: MOVZWL  LOCAL_DESC, R1
                                51      02 02 C6 00054 DIVL2  #2, R1
                                58 AE      51 B0 00057 MOVW   R1, LOCAL_DESC
                                5A AE      02 82 0005B SUBB2  #2, LOCAL_DESC+2
                                5B AE      01 90 0005F MOVB   #1, LOCAL_DESC+3
                                2277
                                2288
                                2291
                                2292
                                2294
                                2296
                                2298
                                2299
                                2300
```

		0167	C7	9F	00063	PUSHAB	P.ABF	2301
			01	DD	00067	PUSHL	#1	
			57	DD	00069	PUSHL	R7	
	68		03	FB	0006B	CALLS	#3, DBG\$PRINT	
		58	AE	9F	0006E	PUSHAB	LOCAL_DESC	2302
8B	AF		01	FB	00071	CALLS	#1, DBG\$PRINT_VMS_VALUE	
		0168	C7	9F	00075	PUSHAB	P.ABG	2303
			01	DD	00079	PUSHL	#1	
			57	DD	0007B	PUSHL	R7	
	68		03	FB	0007D	CALLS	#3, DBG\$PRINT	
	50	58	AE	3C	00080	MOVZWL	LOCAL_DESC, R0	2304
5C	AE		50	C0	00084	ADDL2	R0, LOCAL_DESC+4	
		58	AE	9F	00088	PUSHAB	LOCAL_DESC	2305
FF70	CF		01	FB	0008B	CALLS	#1, DBG\$PRINT_VMS_VALUE	
		0169	C7	9F	00090	PUSHAB	P.ABH	2306
			01	DD	00094	PUSHL	#1	
		0236	31	00096	BRW	27\$		
	0A		50	91	00099	CMPB	R0, #10	2309
			57	12	0009C	BNEQ	6\$	
04	AE	5C	BE	56	0009E	CVTFD	@LOCAL_DESC+4, DVALUE	2321
			02	DD	000A3	PUSHL	#2	2322
			01	DD	000A5	PUSHL	#1	
	7E		07	7D	000A7	MOVQ	#7, -(SP)	
		5C	AE	9F	000AA	PUSHAB	BUFFER_DESC	
		18	AE	9F	000AD	PUSHAB	DVALUE	
	6A		06	FB	000B0	CALLS	#6, FOR\$CVT_D_TG	
	0F		50	E8	000B3	BLBS	R0, 4\$	
		016A	C7	9F	000B6	PUSHAB	P.ABI	2331
			01	DD	000BA	PUSHL	#1	
		00028362	8F	DD	000BC	PUSHL	#164706	
			03	FB	000C2	CALLS	#3, LIB\$SIGNAL	
OC	AE	0040	20	3B	000C5	SKPC	#32, #64, TEXT_BUFFER	2337
			51	DD	000CC	MOVL	R1, R3	
	63		20	3A	000CF	LOCC	#32, LENGTH, (DIGITS)	2338
			51	DD	000D3	MOVL	R1, R2	
		0161	C7	A2	000D6	CMPL	-4(SPACES), EXP_ZERO	2339
			03	12	000DC	BNEQ	5\$	
		52	04	C2	000DE	SUBL2	#4, SPACES	
		01	6C	91	000E1	CMPB	(AP), #1	2340
			6B	1B	000E4	BLEQU	10\$	
	67	08	AC	E9	000E6	BLBC	8(AP), 10\$	
	2D		63	91	000EA	CMPB	(DIGITS), #45	2341
			62	13	000ED	BEQL	10\$	
		0188	C7	9F	000EF	PUSHAB	P.ABJ	2342
			55	11	000F3	BRB	9\$	
	0B		50	91	000F5	CMPB	R0, #11	2346
			60	12	000F8	BNEQ	11\$	
			02	DD	000FA	PUSHL	#2	2354
			01	DD	000FC	PUSHL	#1	
	7E		10	7D	000FE	MOVQ	#16, -(SP)	
		5C	AE	9F	00101	PUSHAB	BUFFER_DESC	
		70	AE	DD	00104	PUSHL	LOCAL_DESC+4	
	6A		06	FB	00107	CALLS	#6, FOR\$CVT_D_TG	
	0F		50	E8	0010A	BLBS	R0, 7\$	
		0189	C7	9F	0010D	PUSHAB	P.ABK	2363
			01	DD	00111	PUSHL	#1	
		00028362	8F	DD	00113	PUSHL	#164706	

OC	AE	0040	69	03	FB	00119	CALLS	#3, LIBSSIGNAL	2369
			8F	20	3B	0011C	SKPC	#32, #64, TEXT_BUFFER	
	63		53	51	DO	00123	MOVL	R1, R3	2370
			50	20	3A	00126	LOCC	#32, LENGTH, (DIGITS)	
		0161	52	51	DO	0012A	MOVL	R1, R2	2371
			C7	A2	D1	0012D	CMPL	-4(SPACES), EXP_ZERO	
				03	12	00133	BNEQ	8\$	
			52	04	C2	00135	SUBL2	#4, SPACES	
			01	6C	91	00138	CMPB	(AP), #1	2372
				14	1B	0013B	BLEQU	10\$	
			10	AC	E9	0013D	BLBC	8(AP), 10\$	
			2D	63	91	00141	CMPB	(DIGITS), #45	2373
				0B	13	00144	BEQL	10\$	
				01A7	C7	9F	PUSHAB	P.ABL	2374
					01	DD	PUSHL	#1	
					57	DD	PUSHL	R7	
			68	03	FB	0014E	CALLS	#3, DBGSPRINT	
				53	DD	00151	PUSHL	DIGITS	2375
	7E		52	53	C3	00153	SUBL3	DIGITS, SPACES, -(SP)	
				0175	31	00157	BRW	27\$	
			1B	50	91	0015A	CMPB	R0, #27	2378
				57	12	0015D	BNEQ	14\$	
				03	DD	0015F	PUSHL	#3	2386
				01	DD	00161	PUSHL	#1	
			7E	0F	7D	00163	MOVQ	#15, -(SP)	
				5C	AE	9F	PUSHAB	BUFFER_DESC	
				70	AE	DD	PUSHL	LOCAL_DESC+4	
		00000000G	00	06	FB	0016C	CALLS	#6, FOR\$CVT_G_TG	
			0F	50	E8	00173	BLBS	R0, 12\$	
				01A8	C7	9F	PUSHAB	P.ABM	2395
					01	DD	PUSHL	#1	
				00028362	8F	DD	PUSHL	#164706	
			69	03	FB	00182	CALLS	#3, LIBSSIGNAL	
OC	AE	0040	8F	20	3B	00185	SKPC	#32, #64, TEXT_BUFFER	2401
			55	51	DO	0018C	MOVL	R1, R5	
	65		50	20	3A	0018F	LOCC	#32, LENGTH, (DIGITS)	2402
			54	51	DO	00193	MOVL	R1, R4	
0161	C7	FB	A4	05	29	00196	CMPC3	#5, -5(SPACES), EXP_ZERO	2403
				03	12	0019D	BNEQ	13\$	
			54	05	C2	0019F	SUBL2	#5, SPACES	
			01	6C	91	001A2	CMPB	(AP), #1	2404
				70	1B	001A5	BLEQU	18\$	
			6C	AC	E9	001A7	BLBC	8(AP), 18\$	
			2D	65	91	001AB	CMPB	(DIGITS), #45	2405
				67	13	001AE	BEQL	18\$	
				01C6	C7	9F	PUSHAB	P.ABN	2406
					5A	11	BRB	17\$	
			1C	50	91	001B6	CMPB	R0, #28	2410
				65	12	001B9	BNEQ	19\$	
				04	DD	001BB	PUSHL	#4	2418
				01	DD	001BD	PUSHL	#1	
			7E	21	7D	001BF	MOVQ	#33, -(SP)	
				5C	AE	9F	PUSHAB	BUFFER_DESC	
				70	AE	DD	PUSHL	LOCAL_DESC+4	
		00000000G	00	06	FB	001C8	CALLS	#6, FOR\$CVT_H_TG	
			0F	50	E8	001CF	BLBS	R0, 15\$	
				01C7	C7	9F	PUSHAB	P.ABO	2427

			00028362	01	DD	001D6	PUSHL	#1		
				8F	DD	001D8	PUSHL	#164706		
OC	AE	0040	69	03	FB	001DE	CALLS	#3, LIB\$SIGNAL		
			8F	20	3B	001E1	158:	SKPC	#32, #64, TEXT_BUFFER	2433
	65		55	51	DD	001E8	MOVL	R1, R5		
			50	20	3A	001EB	LOCC	#32, LENGTH, (DIGITS)		2434
0161	C7	FA	54	51	DD	001EF	MOVL	R1, R4		
			A4	06	29	001F2	CMPC3	#6, -6(SPACES), EXP_ZERO		2435
				03	12	001F9	BNEQ	16\$		
			54	06	C2	001FB	SUBL2	#6, SPACES		
			01	6C	91	001FE	16\$:	CMPB	(AP), #1	2436
				14	1B	00201	BLEQU	18\$		
		08	10	AC	E9	00203	BLBC	8(AP), 18\$		
			2D	65	91	00207	CMPB	(DIGITS), #45		2437
				0B	13	0020A	BEQL	18\$		
		01E5		C7	9F	0020C	PUSHAB	P.ABP		2438
				01	DD	00210	17\$:	PUSHL	#1	
				57	DD	00212	PUSHL	R7		
			68	03	FB	00214	CALLS	#3, DBG\$PRINT		
				55	DD	00217	18\$:	PUSHL	DIGITS	2439
7E			54	55	C3	00219	SUBL3	DIGITS, SPACES, -(SP)		
				00AF	31	0021D	BRW	27\$		
			0E	50	91	00220	19\$:	CMPB	R0, #14	2442
				3C	12	00223	BNEQ	22\$		
		58	50	AE	3C	00225	MOVZWL	LOCAL_DESC, R0		2450
	0800		8F	50	B1	00229	CMPW	R0, #2048		
				05	1B	0022E	BLEQU	20\$		
		0800	50	8F	3C	00230	MOVZWL	#2048, R0		
	58		AE	50	B0	00235	20\$:	MOVW	R0, LOCAL_DESC	
		5C	51	AE	DD	00239	MOVL	LOCAL_DESC+4, ADDR		2462
		58	50	AE	3C	0023D	MOVZWL	LOCAL_DESC, BYTES		2463
61			50	00	0C	00241	PROBER	#0, BYTES, (ADDR)		2464
				0D	12	00245	BNEQ	21\$		
				51	DD	00247	PUSHL	ADDR		2466
				01	DD	00249	PUSHL	#1		
		00028228		8F	DD	0024B	PUSHL	#164392		
			69	03	FB	00251	CALLS	#3, LIB\$SIGNAL		
	5C			AE	DD	00254	21\$:	PUSHL	LOCAL_DESC+4	2469
	7E	5C		AE	3C	00257	MOVZWL	LOCAL_DESC, -(SP)		2468
		01E6		C7	9F	0025B	PUSHAB	P.ABP		
				70	11	0025F	BRB	28\$		
		00000000G	01	00	91	00261	22\$:	CMPB	DBG\$GB_LANGUAGE, #1	2481
				22	12	00268	BNEQ	25\$		
		5A	02	AE	91	0026A	CMPB	LOCAL_DESC+2, #2		2483
				06	12	0026E	BNEQ	23\$		
5A	AE			06	90	00270	MOVB	#6, LOCAL_DESC+2		2484
				16	11	00274	BRB	25\$		
		5A	03	AE	91	00276	23\$:	CMPB	LOCAL_DESC+2, #3	2485
				06	12	0027A	BNEQ	24\$		
5A	AE			07	90	0027C	MOVB	#7, LOCAL_DESC+2		2486
				0A	11	00280	BRB	25\$		
		5A	04	AE	91	00282	24\$:	CMPB	LOCAL_DESC+2, #4	2487
				04	12	00286	BNEQ	25\$		
5A	AE			08	90	00288	MOVB	#8, LOCAL_DESC+2		2488
				5E	DD	0028C	25\$:	PUSHL	SP	2490
		50		AE	9F	0028E	PUSHAB	BUFFER_DESC		
		60		AE	9F	00291	PUSHAB	LOCAL_DESC		

		00000000G	00		03	FB	00294		CALLS	#3, DBG\$CVT DX DX		
			50	5A	AE	9A	0029B		MOVZBL	LOCAL_DESC+2, R0		2492
21		00000000'	EF		50	E1	0029F		BBC	R0, SIGNED_DTYPE, 26\$		
			01		6C	91	002A7		CMPB	(AP), #1		2493
					1C	1B	002AA		BLEQU	26\$		
			18	0B	AC	E9	002AC		BLBC	8(AP), 26\$		
			2D	0C	AE	91	002B0		CMPB	TEXT_BUFFER, #45		2494
					12	13	002B4		BEQL	26\$		
			2B	0C	AE	91	002B6		CMPB	TEXT_BUFFER, #43		
					0C	13	002BA		BEQL	26\$		
0D	AE		OC	AE	6E	28	002BC		MOVC3	TEXT_LENGTH, TEXT_BUFFER, TEXT_BUFFER+1		2496
			OC	AE	2B	90	002C2		MOVB	#43, TEXT_BUFFER		2497
					6E	B6	002C6		INCW	TEXT_LENGTH		2498
				0C	AE	9F	002C8	26\$:	PUSHAB	TEXT_BUFFER		2501
			7E	04	AE	3C	002CB		MOVZWL	TEXT_LENGTH, -(SP)		
					57	DD	002CF	27\$:	PUSHL	R7		
			6B		03	FB	002D1	28\$:	CALLS	#3, DBG\$PRINT		
					04	002D4			RET			2504

; Routine Size: 725 bytes, Routine Base: DBG\$CODE + 1623

DBGVALUES
V04-000

K 1
16-Sep-1984 02:45:26
14-Sep-1984 12:17:54

VAX-11 Bliss-32 V4.0-742
[DEBUG.SRC]DBGVALUES.B32;1

Page 93
(29)

: 2400 2505 1 END
: 2401 2506 0 ELUDOM

.EXTRN LIB\$SIGNAL

PSECT SUMMARY

Name	Bytes	Attributes
DBG\$OWN	6	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
DBG\$PLIT	496	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)
DBG\$CODE	6392	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	55	0	1000	00:01.9
_\$255\$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1	32	0	0	7	00:00.1
_\$255\$DUA28:[DEBUG.OBJ]DBGLIB.L32;1	1545	191	12	97	00:01.9
_\$255\$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1	418	15	3	31	00:00.3
_\$255\$DUA28:[DEBUG.OBJ]DBGMSG.L32;1	386	10	2	22	00:00.3

COMMAND QUALIFIERS

: BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:DBGVALUES/OBJ=OBJ\$:DBGVALUES MSRC\$:DBGVALUES/UPDATE=(ENH\$:DBGVALUES)

: Size: 6392 code + 502 data bytes
: Run Time: 01:43.8
: Elapsed Time: 01:53.6
: Lines/CPU Min: 1448
: Lexemes/CPU-Min: 18095
: Memory Used: 478 pages
: Compilation Complete

0096 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

0097 AH-BT13A-SE
VAX/VMS V4.0

**DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY**